Exhibit A

US007860254B2

(12) **United States Patent**
Soliman

(10) **Patent No.: US 7,860,254 B2**
(45) **Date of Patent: Dec. 28, 2010**

(54) **COMPUTER SYSTEM SECURITY VIA DYNAMIC ENCRYPTION**

(76) Inventor: **Hamdy Soliman**, 1214 Calle De Lago, Socorro, NM (US) 87801

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 2596 days.

(21) Appl. No.: **10/387,711**

(22) Filed: **Mar. 13, 2003**

(65) **Prior Publication Data**

US 2004/0179682 A1 Sep. 16, 2004

(51) **Int. Cl.**
H04L 9/16 (2006.01)
H04L 9/00 (2006.01)
H04L 29/06 (2006.01)
H04L 9/08 (2006.01)

(52) **U.S. Cl.** **380/279**; 380/46; 713/155

(58) **Field of Classification Search** 380/46, 380/279; 713/155
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,568,966	A *	2/1986	Lippel	....	386/21
5,091,942	A *	2/1992	Dent	....	380/46
5,253,294	A *	10/1993	Maurer	....	380/264
5,365,588	A *	11/1994	Bianco et al.	....	380/42
5,491,750	A *	2/1996	Bellare et al.	....	713/155
6,301,664	B1 *	10/2001	Di-Crescenzo et al.	......	713/189
6,839,434	B1 *	1/2005	Mizikovsky	....	380/247
2003/0217292	A1 *	11/2003	Steiger et al.	....	713/201

OTHER PUBLICATIONS

Menezes et al. "Handbook of Applied Cryptography" 1997, CRC Press LLC, pp. 397-403.*
Schneier, Bruce "Applied Cryptography Second Edition" 1996, John Wiley and Sons, Inc.; pp. 197-211.*
Silberschatz et al. "Operating System Concepts" 5th Edition 1999, John Wiley and Sons, pp. 63 and 99.*

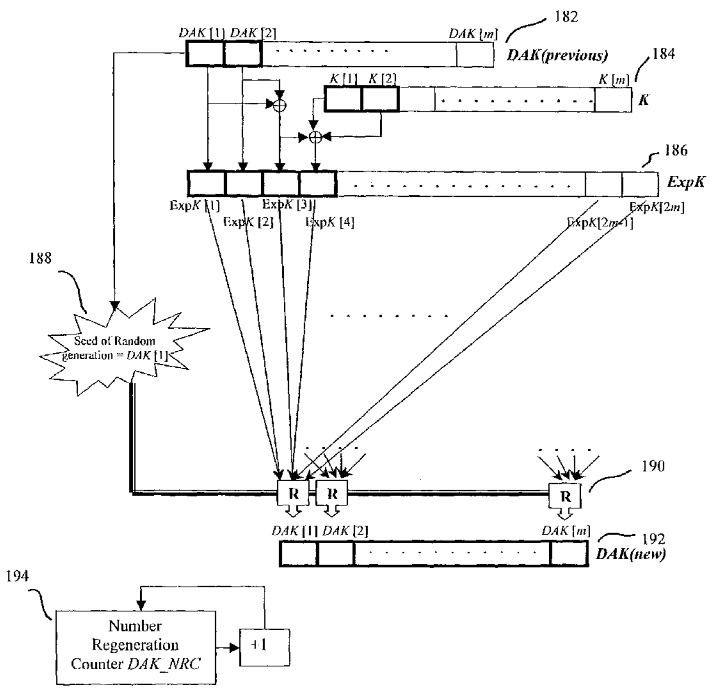
* cited by examiner

Primary Examiner—Matthew T Henning

(57) **ABSTRACT**

A dynamic computer system security method and system using dynamic encryption and full synchronization between system nodes. A data record from a data stream created by a source user is encrypted with an initial dynamic session key. A new dynamic session key is generated based upon a data record and a previous dynamic session key. The new dynamic session key is then used to encrypt the next data record. A central authority is used to synchronize and authenticate both source and destination users with dynamic authentication keys. The central authority and users constantly regenerate new dynamic authentication keys. A child process is forked to ensure synchronization and authentication of dynamic authentication keys of each node upon a request for a secure communication establishment from a user. The central authority generates the initial dynamic session key with the current dynamic authentication key to begin a secure communication session.

15 Claims, 16 Drawing Sheets



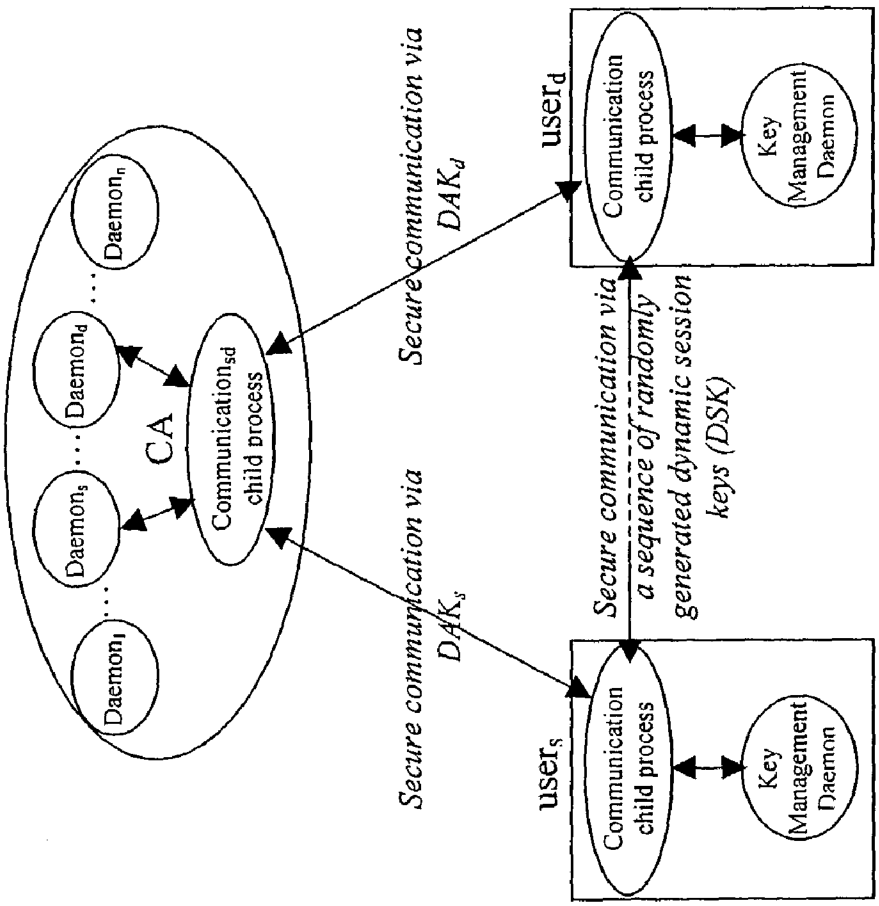

CA
Daemon$_1$
Daemon$_s$
Daemon$_d$
Daemon$_n$
Communication$_{sd}$ child process
Secure communication via DAK_s
Secure communication via DAK_d
user$_s$
user$_d$
Communication child process
Key Management Daemon
Communication child process
Key Management Daemon
Secure communication via a sequence of randomly generated dynamic session keys (DSK)

FIG. 1b

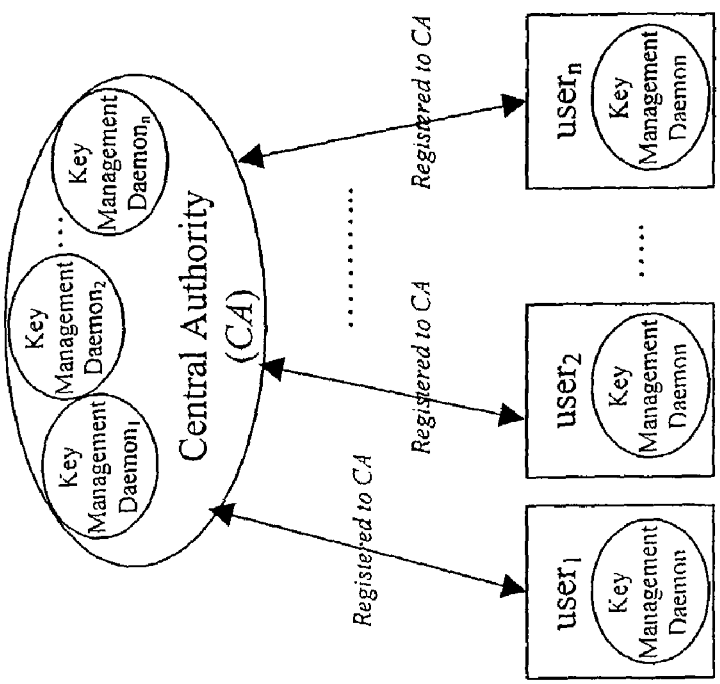

Central Authority (CA)
Key Management Daemon$_1$
Key Management Daemon$_2$
Key Management Daemon$_n$
Registered to CA
Registered to CA
Registered to CA
user$_1$
Key Management Daemon
user$_2$
Key Management Daemon
user$_n$
Key Management Daemon

FIG. 1a

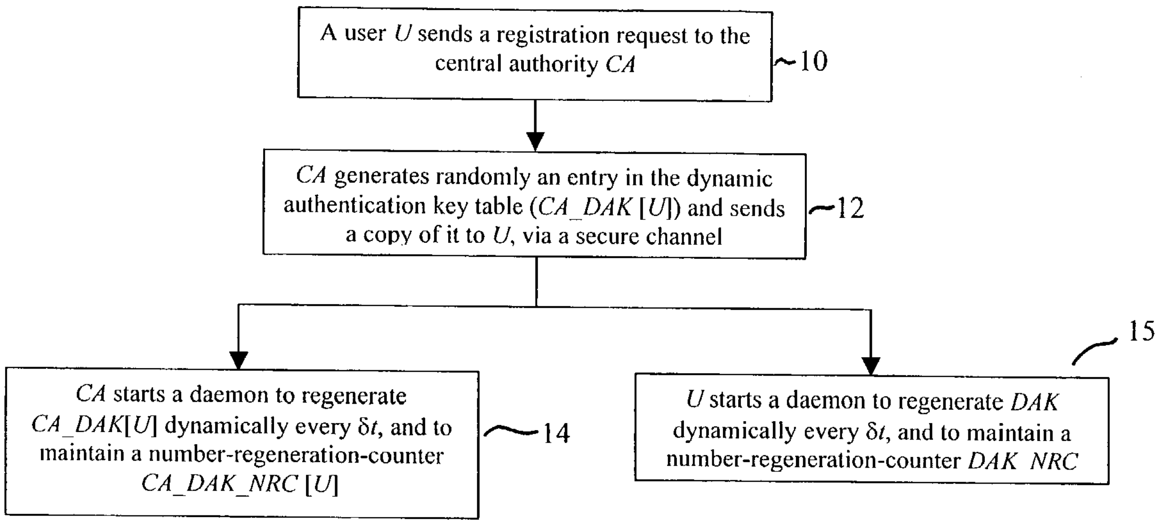
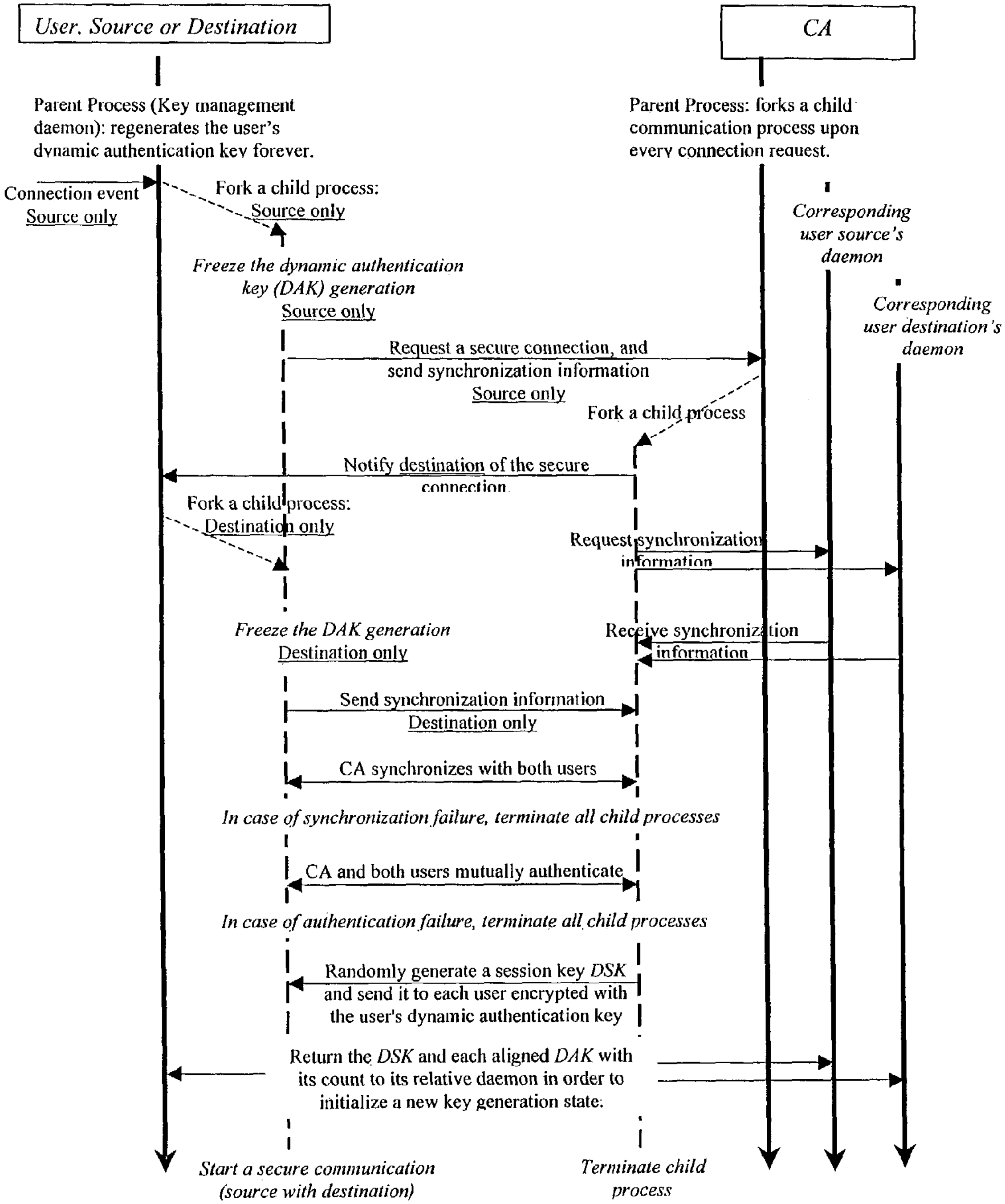
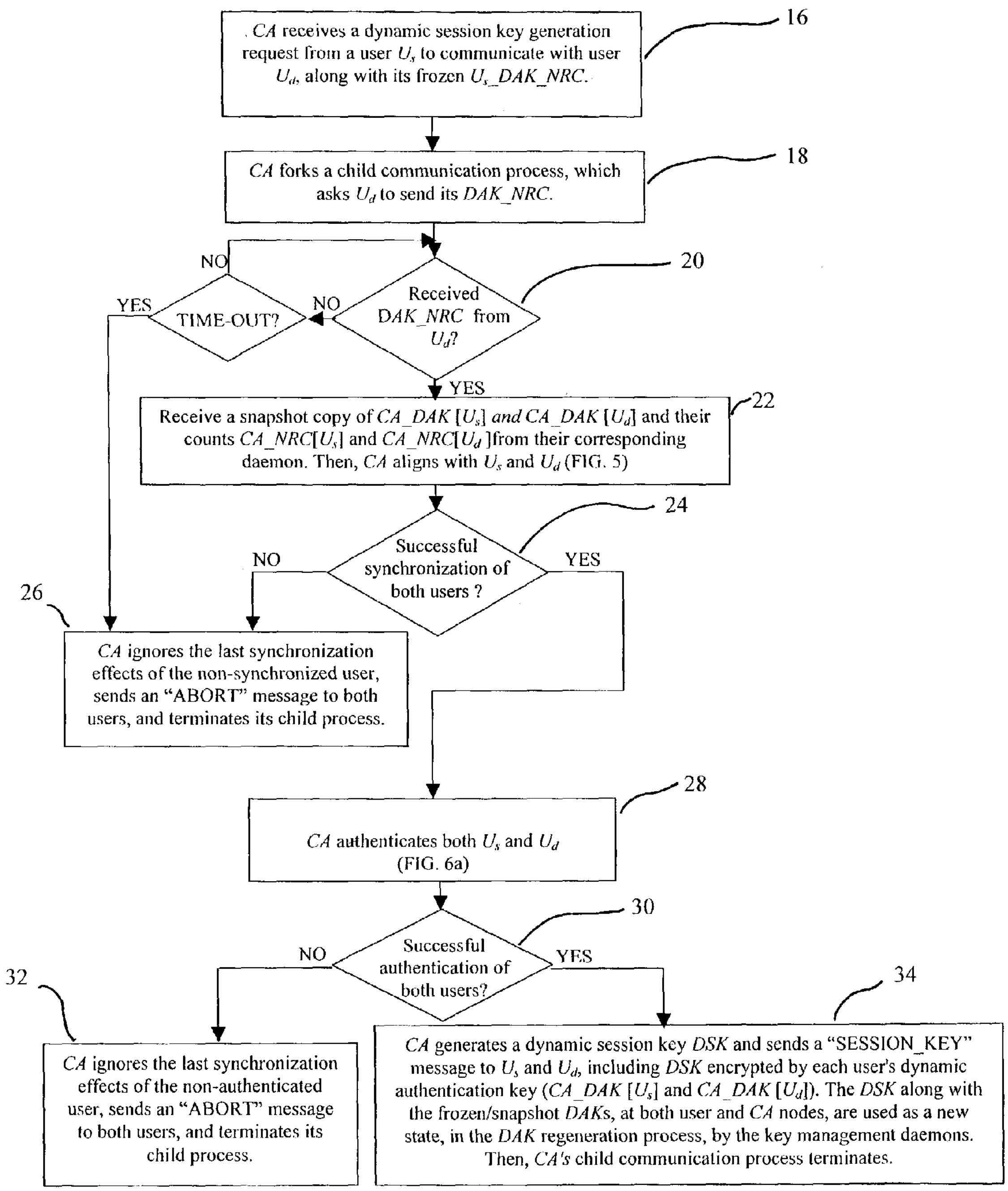

A user U sends a registration request to the central authority CA
10
CA generates randomly an entry in the dynamic authentication key table (CA_DAK [U]) and sends a copy of it to U, via a secure channel
12
CA starts a daemon to regenerate CA_DAK[U] dynamically every δt, and to maintain a number-regeneration-counter CA_DAK_NRC [U]
14
U starts a daemon to regenerate DAK dynamically every δt, and to maintain a number-regeneration-counter DAK NRC
15

FIG. 2



User, Source or Destination
CA
Parent Process (Key management daemon): regenerates the user's dynamic authentication key forever.
Parent Process: forks a child communication process upon every connection request.
Connection event
Source only
Fork a child process:
Source only
Corresponding user source's daemon
Freeze the dynamic authentication key (DAK) generation
Source only
Corresponding user destination's daemon
Request a secure connection, and send synchronization information
Source only
Fork a child process
Notify destination of the secure connection
Fork a child process:
Destination only
Request synchronization information
Freeze the DAK generation
Destination only
Receive synchronization information
Send synchronization information
Destination only
CA synchronizes with both users
In case of synchronization failure, terminate all child processes
CA and both users mutually authenticate
In case of authentication failure, terminate all child processes
Randomly generate a session key DSK and send it to each user encrypted with the user's dynamic authentication key
Return the DSK and each aligned DAK with its count to its relative daemon in order to initialize a new key generation state.
Start a secure communication (source with destination)
Terminate child process

FIG. 3



CA receives a dynamic session key generation request from a user U_s to communicate with user U_d, along with its frozen U_s_DAK_NRC.
16
CA forks a child communication process, which asks U_d to send its DAK_NRC.
18
NO
YES
TIME-OUT?
NO
Received DAK_NRC from U_d?
20
YES
Receive a snapshot copy of CA_DAK [U_s] and CA_DAK [U_d] and their counts CA_NRC[U_s] and CA_NRC[U_d]from their corresponding daemon. Then, CA aligns with U_s and U_d (FIG. 5)
22
Successful synchronization of both users ?
24
NO
YES
26
CA ignores the last synchronization effects of the non-synchronized user, sends an "ABORT" message to both users, and terminates its child process.
28
CA authenticates both U_s and U_d (FIG. 6a)
30
Successful authentication of both users?
NO
YES
32
34
CA ignores the last synchronization effects of the non-authenticated user, sends an "ABORT" message to both users, and terminates its child process.
CA generates a dynamic session key DSK and sends a "SESSION_KEY" message to U_s and U_d, including DSK encrypted by each user's dynamic authentication key (CA_DAK [U_s] and CA_DAK [U_d]). The DSK along with the frozen/snapshot DAKs, at both user and CA nodes, are used as a new state, in the DAK regeneration process, by the key management daemons. Then, CA's child communication process terminates.

FIG. 4

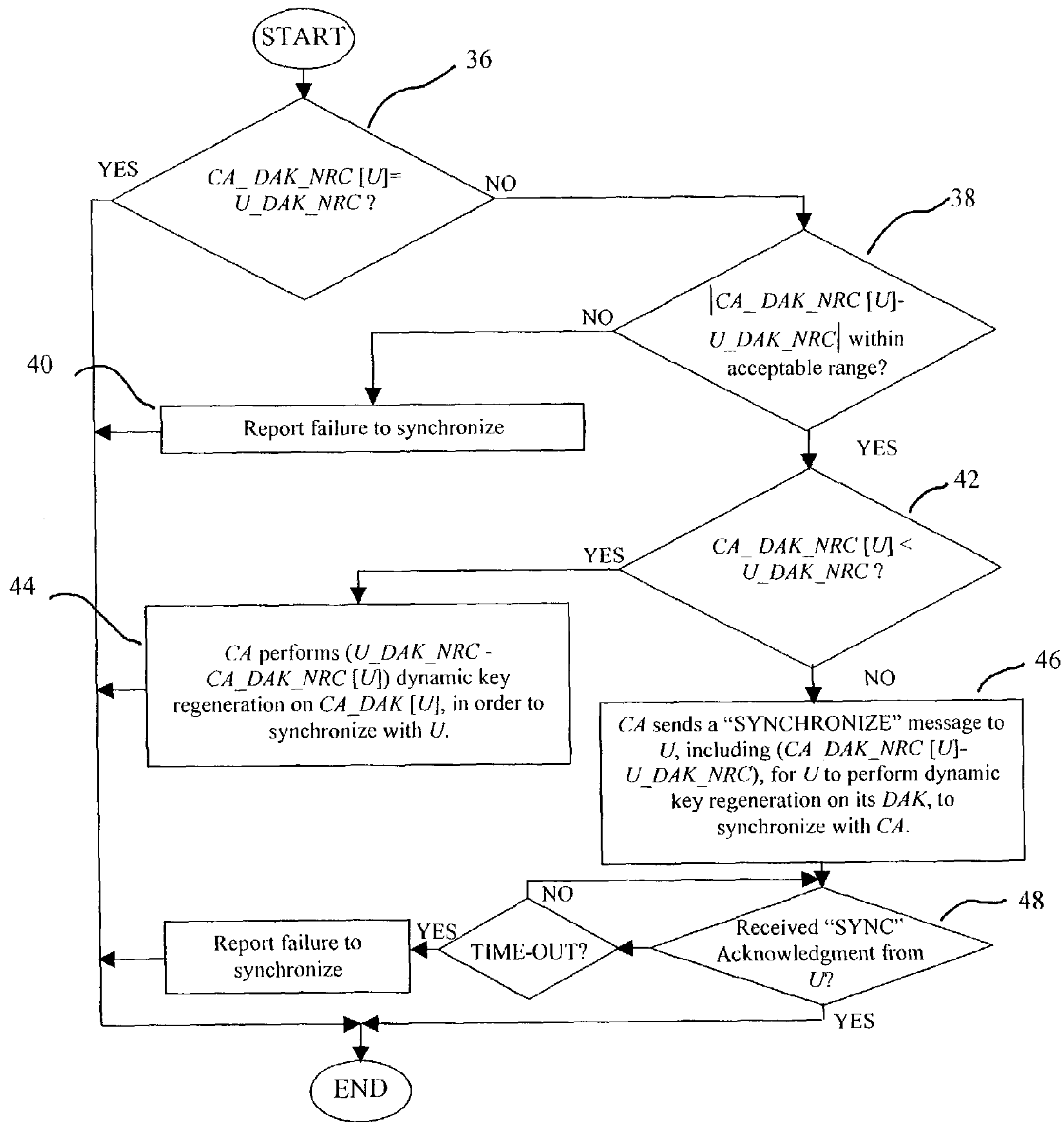

START
36
CA_DAK_NRC [U]= U_DAK_NRC ?
YES
NO
38
\|CA_DAK_NRC [U]- U_DAK_NRC\| within acceptable range?
NO
40
Report failure to synchronize
YES
42
CA_DAK_NRC [U] < U_DAK_NRC ?
YES
44
CA performs (U_DAK_NRC - CA_DAK_NRC [U]) dynamic key regeneration on CA_DAK [U], in order to synchronize with U.
NO
46
CA sends a "SYNCHRONIZE" message to U, including (CA_DAK_NRC [U]- U_DAK_NRC), for U to perform dynamic key regeneration on its DAK, to synchronize with CA.
NO
48
Received "SYNC" Acknowledgment from U?
TIME-OUT?
YES
Report failure to synchronize
YES
END

FIG. 5

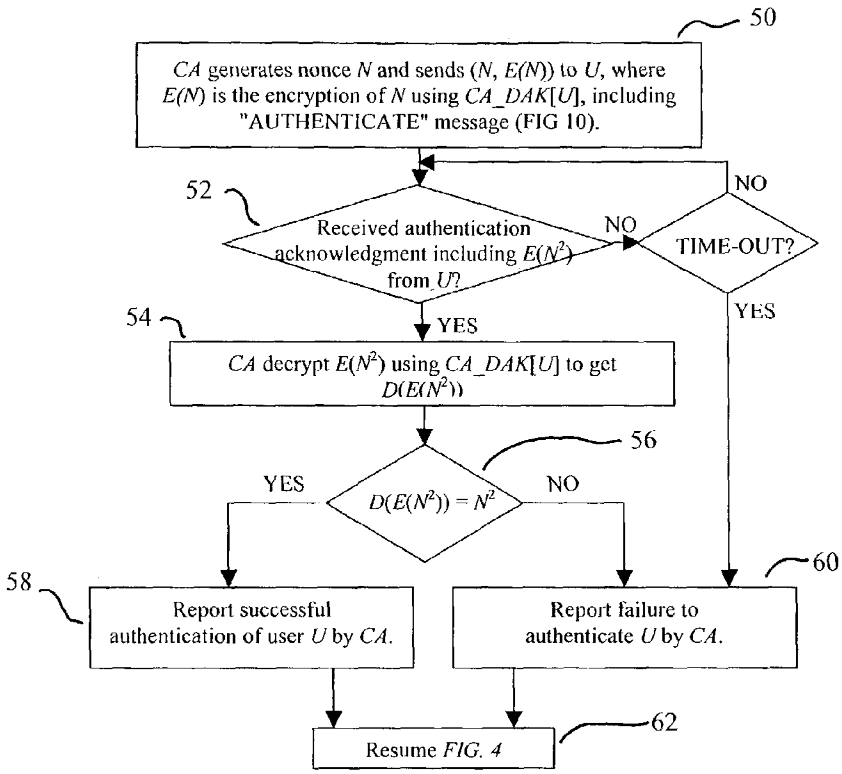

50
CA generates nonce N and sends (N, E(N)) to U, where E(N) is the encryption of N using CA_DAK[U], including "AUTHENTICATE" message (FIG 10).
52
Received authentication acknowledgment including E(N²) from U?
NO
TIME-OUT?
NO
YES
54
YES
CA decrypt E(N²) using CA_DAK[U] to get D(E(N²))
56
YES
D(E(N²)) = N²
NO
58
Report successful authentication of user U by CA.
60
Report failure to authenticate U by CA.
Resume FIG. 4
62

FIG. 6a

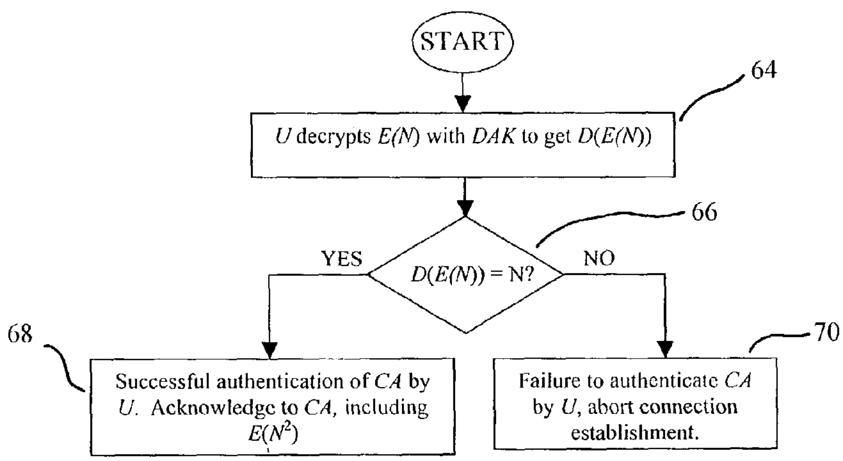

START
64
U decrypts E(N) with DAK to get D(E(N))
66
YES
D(E(N)) = N?
NO
68
Successful authentication of CA by U. Acknowledge to CA, including E(N²)
70
Failure to authenticate CA by U, abort connection establishment.

FIG. 6b

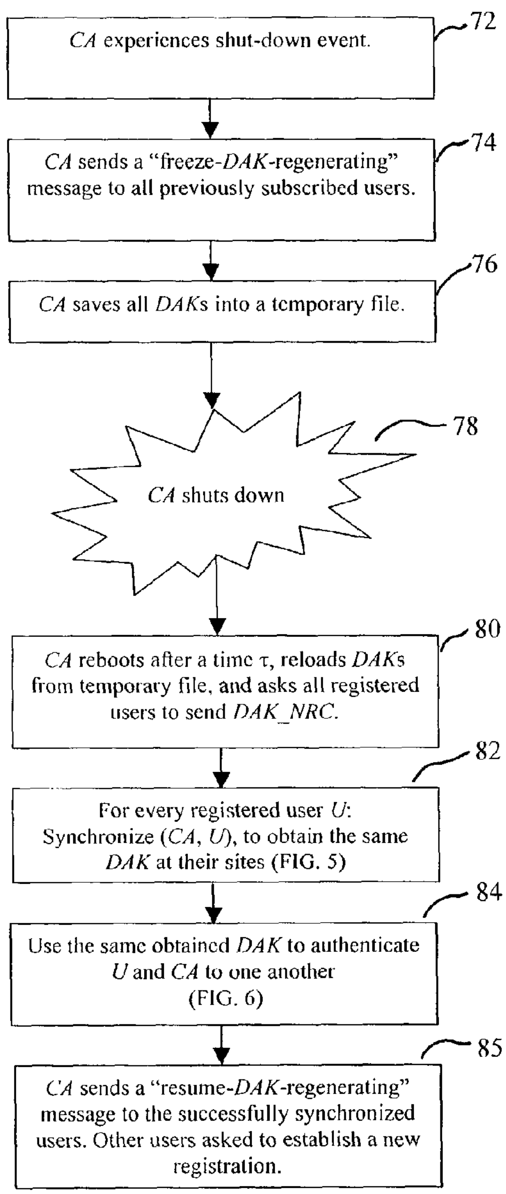

CA experiences shut-down event.
72
CA sends a "freeze-DAK-regenerating" message to all previously subscribed users.
74
CA saves all DAKs into a temporary file.
76
CA shuts down
78
CA reboots after a time τ, reloads DAKs from temporary file, and asks all registered users to send DAK_NRC.
80
For every registered user U: Synchronize (CA, U), to obtain the same DAK at their sites (FIG. 5)
82
Use the same obtained DAK to authenticate U and CA to one another (FIG. 6)
84
CA sends a "resume-DAK-regenerating" message to the successfully synchronized users. Other users asked to establish a new registration.
85

FIG. 7a

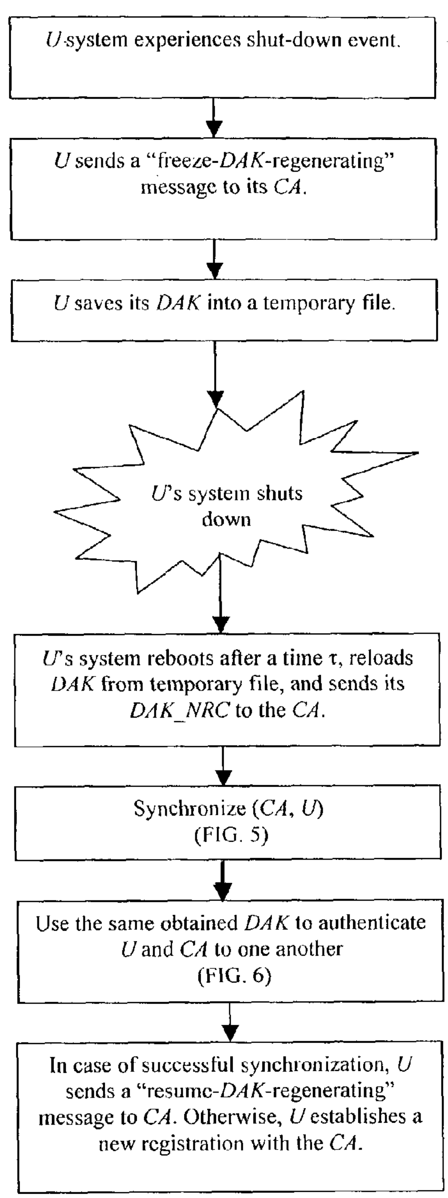

U system experiences shut-down event.
U sends a "freeze-DAK-regenerating" message to its CA.
U saves its DAK into a temporary file.
U's system shuts down
U's system reboots after a time τ, reloads DAK from temporary file, and sends its DAK_NRC to the CA.
Synchronize (CA, U) (FIG. 5)
Use the same obtained DAK to authenticate U and CA to one another (FIG. 6)
In case of successful synchronization, U sends a "resume-DAK-regenerating" message to CA. Otherwise, U establishes a new registration with the CA.

FIG. 7b

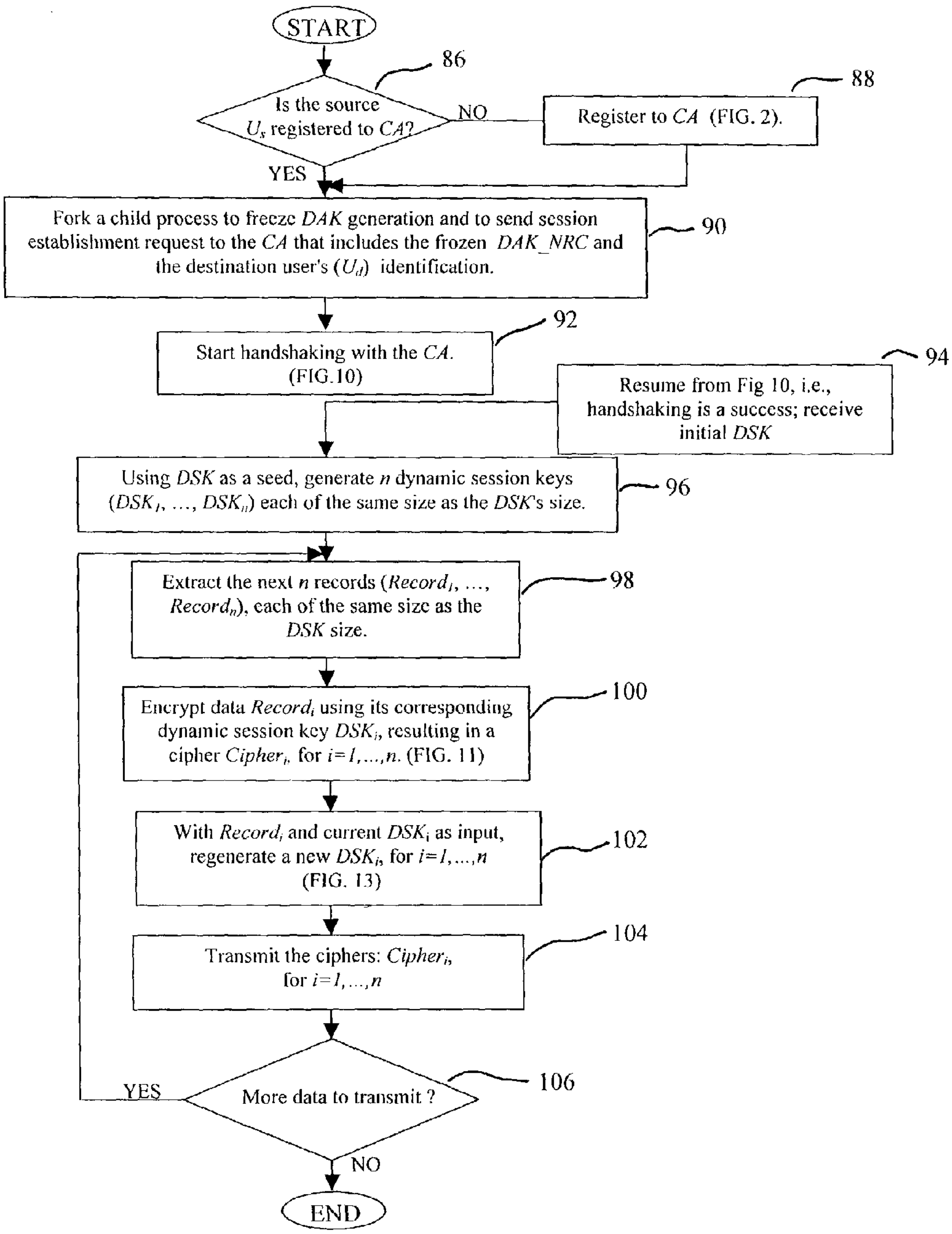
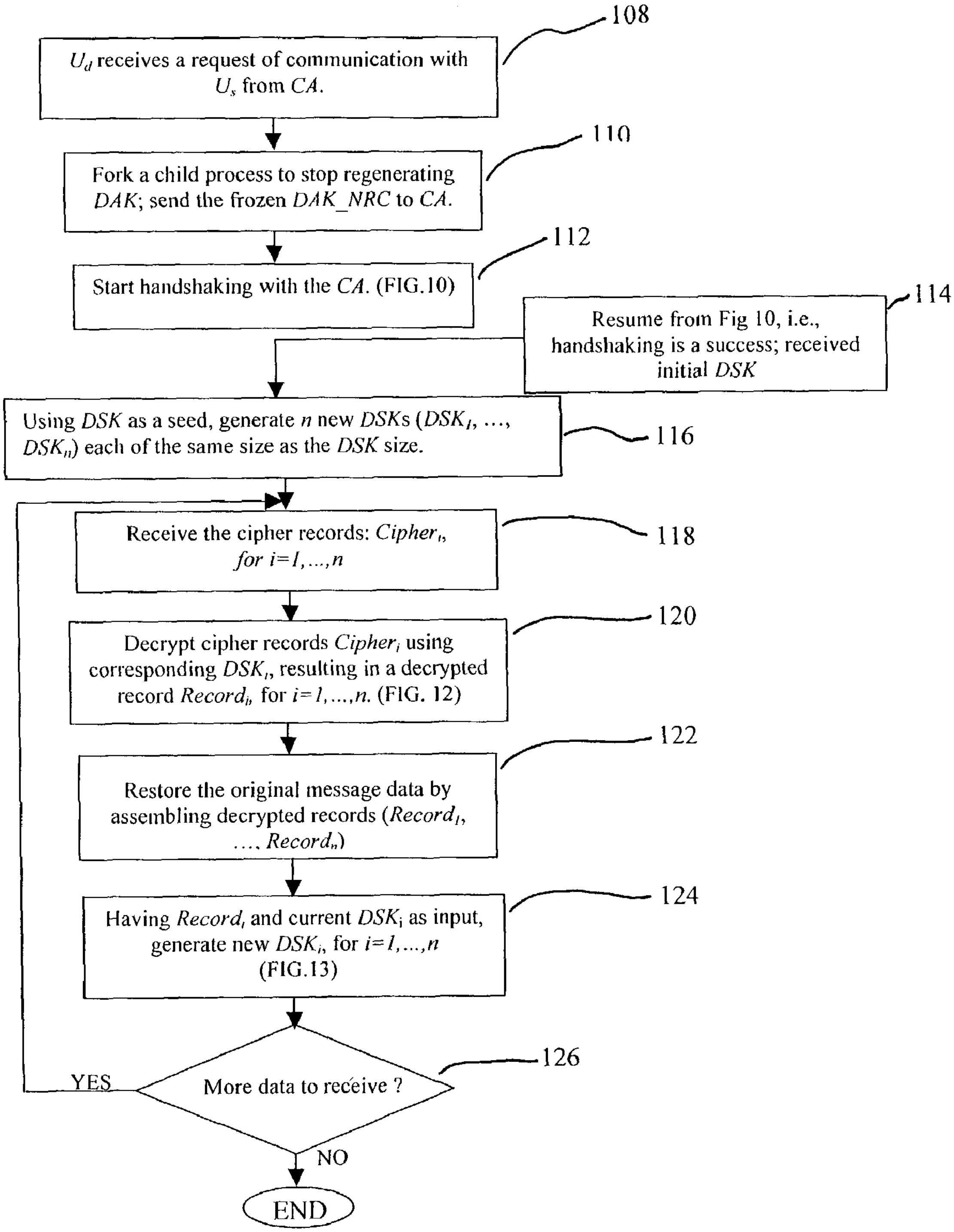

START
86
Is the source U_s registered to CA?
NO
Register to CA (FIG. 2).
88
YES
Fork a child process to freeze DAK generation and to send session establishment request to the CA that includes the frozen DAK_NRC and the destination user's (U_d) identification.
90
Start handshaking with the CA. (FIG.10)
92
Resume from Fig 10, i.e., handshaking is a success; receive initial DSK
94
Using DSK as a seed, generate n dynamic session keys (DSK_1, ..., DSK_n) each of the same size as the DSK's size.
96
Extract the next n records ($Record_1$, ..., $Record_n$), each of the same size as the DSK size.
98
Encrypt data $Record_i$ using its corresponding dynamic session key DSK_i, resulting in a cipher $Cipher_i$, for $i=1,...,n$. (FIG. 11)
100
With $Record_i$ and current DSK_i as input, regenerate a new DSK_i, for $i=1,...,n$ (FIG. 13)
102
Transmit the ciphers: $Cipher_i$, for $i=1,...,n$
104
YES
More data to transmit ?
106
NO
END

FIG. 8



108
U_d receives a request of communication with U_s from CA.
110
Fork a child process to stop regenerating DAK; send the frozen DAK_NRC to CA.
112
Start handshaking with the CA. (FIG.10)
114
Resume from Fig 10, i.e., handshaking is a success; received initial DSK
Using DSK as a seed, generate n new DSKs (DSK_1, …, DSK_n) each of the same size as the DSK size.
116
Receive the cipher records: $Cipher_i$, $for\ i=1,\ldots,n$
118
Decrypt cipher records $Cipher_i$ using corresponding DSK_i, resulting in a decrypted record $Record_i$, for $i=1,\ldots,n$. (FIG. 12)
120
Restore the original message data by assembling decrypted records ($Record_1$, …, $Record_n$)
122
Having $Record_i$ and current DSK_i as input, generate new DSK_i, for $i=1,\ldots,n$ (FIG.13)
124
More data to receive ?
126
YES
NO
END

FIG. 9

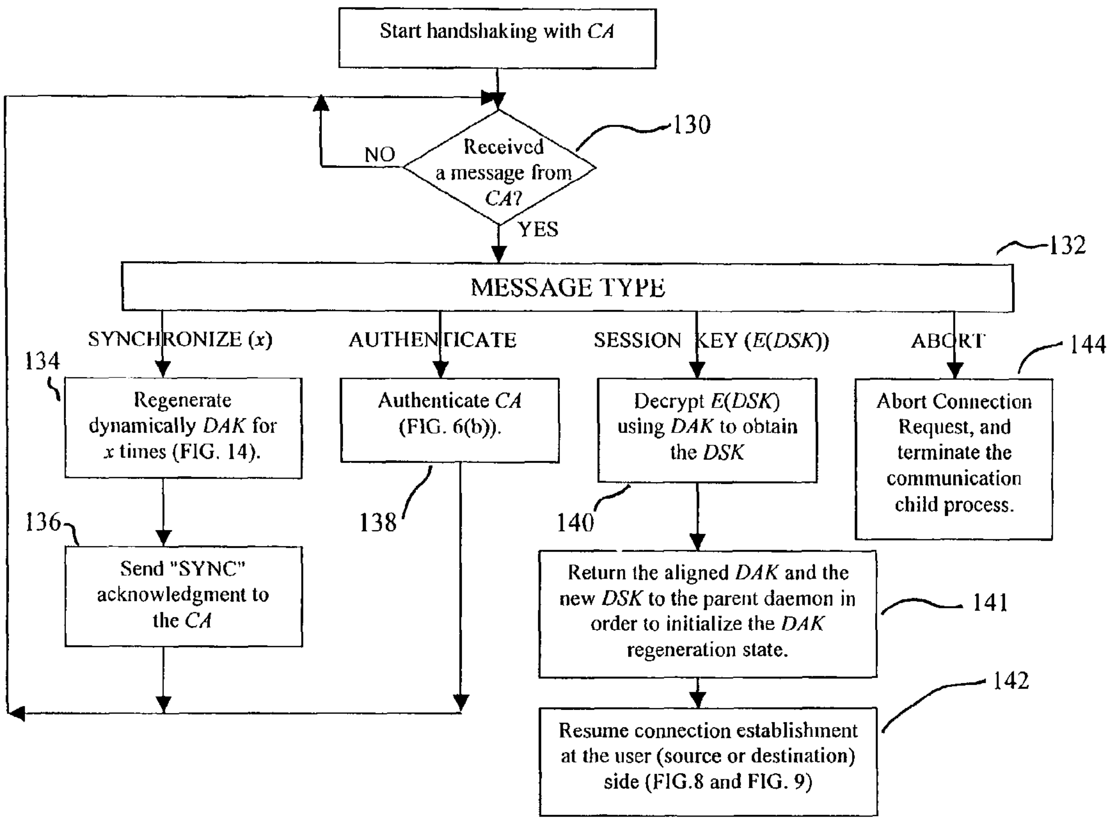

Start handshaking with CA
130
Received a message from CA?
NO
YES
132
MESSAGE TYPE
SYNCHRONIZE (x)
AUTHENTICATE
SESSION KEY (E(DSK))
ABORT
134
Regenerate dynamically DAK for x times (FIG. 14).
Authenticate CA (FIG. 6(b)).
Decrypt E(DSK) using DAK to obtain the DSK
144
Abort Connection Request, and terminate the communication child process.
136
Send "SYNC" acknowledgment to the CA
138
140
Return the aligned DAK and the new DSK to the parent daemon in order to initialize the DAK regeneration state.
141
142
Resume connection establishment at the user (source or destination) side (FIG.8 and FIG. 9)

FIG. 10

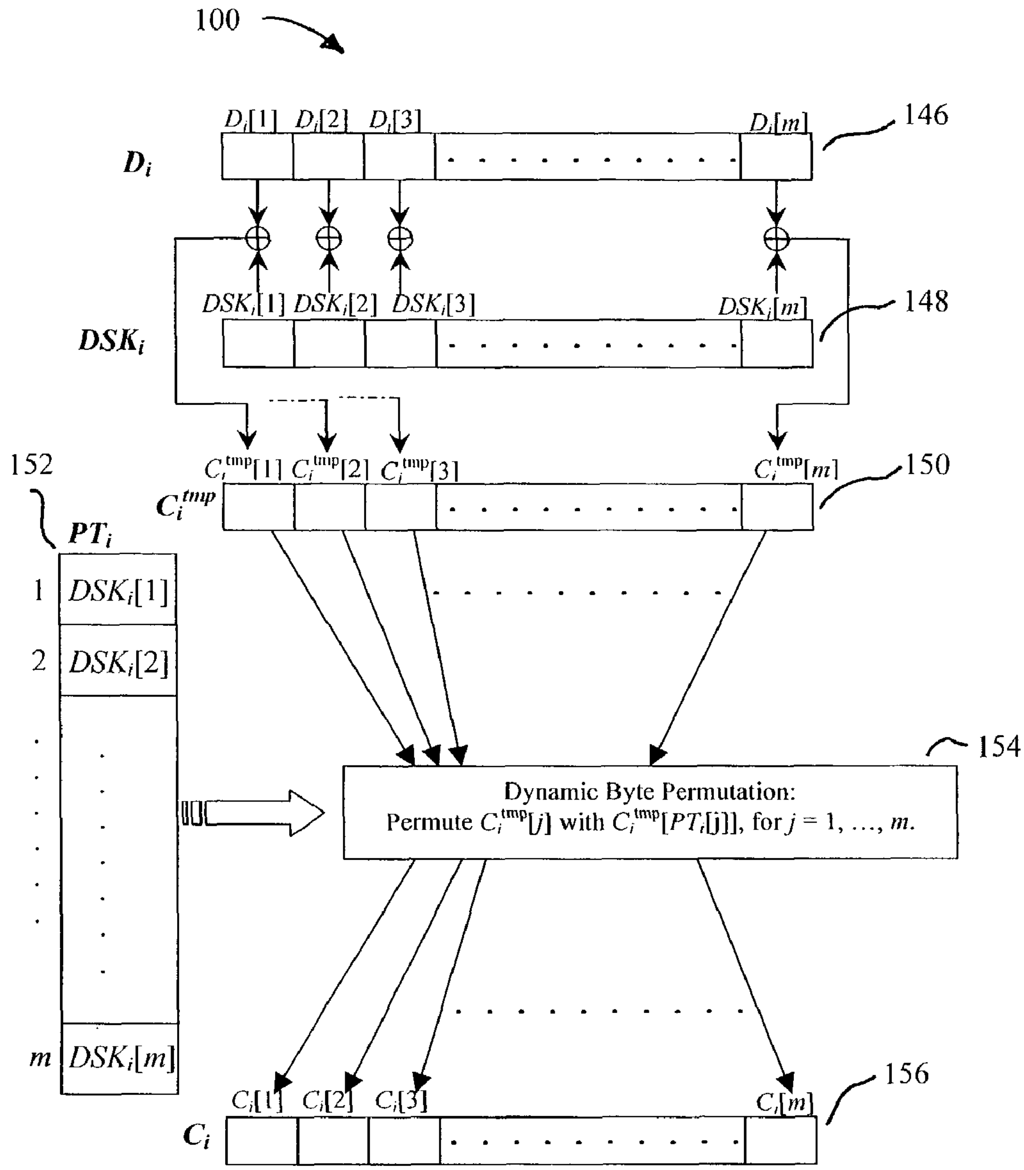

100
$D_i[1]$ $D_i[2]$ $D_i[3]$ $D_i[m]$
146
D_i
$DSK_i[1]$ $DSK_i[2]$ $DSK_i[3]$ $DSK_i[m]$
148
DSK_i
152
$C_i^{tmp}[1]$ $C_i^{tmp}[2]$ $C_i^{tmp}[3]$ $C_i^{tmp}[m]$
150
C_i^{tmp}
PT_i
1 $DSK_i[1]$
2 $DSK_i[2]$
m $DSK_i[m]$
154
Dynamic Byte Permutation:
Permute $C_i^{tmp}[j]$ with $C_i^{tmp}[PT_i[j]]$, for $j = 1, \ldots, m$.
$C_i[1]$ $C_i[2]$ $C_i[3]$ $C_i[m]$
156
C_i

FIG. 11

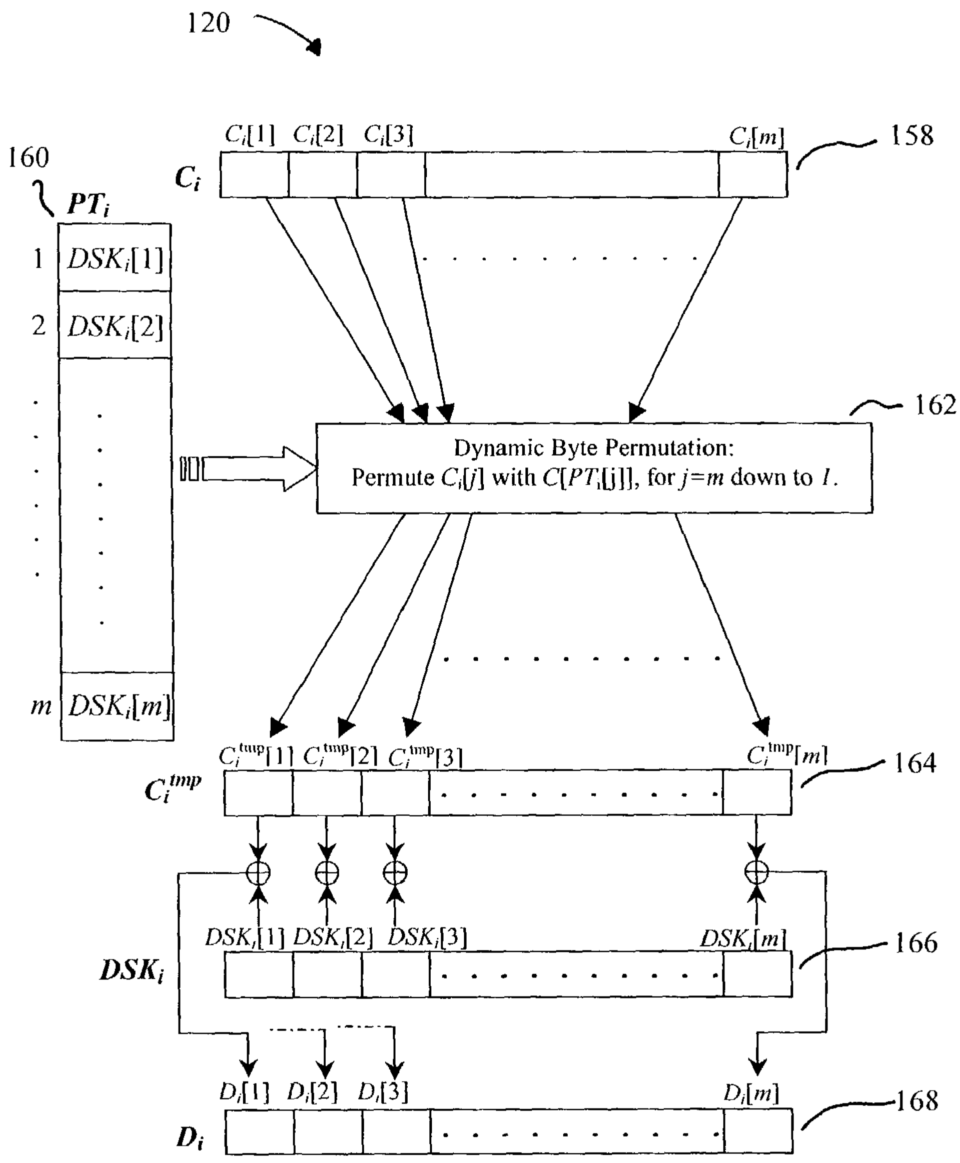
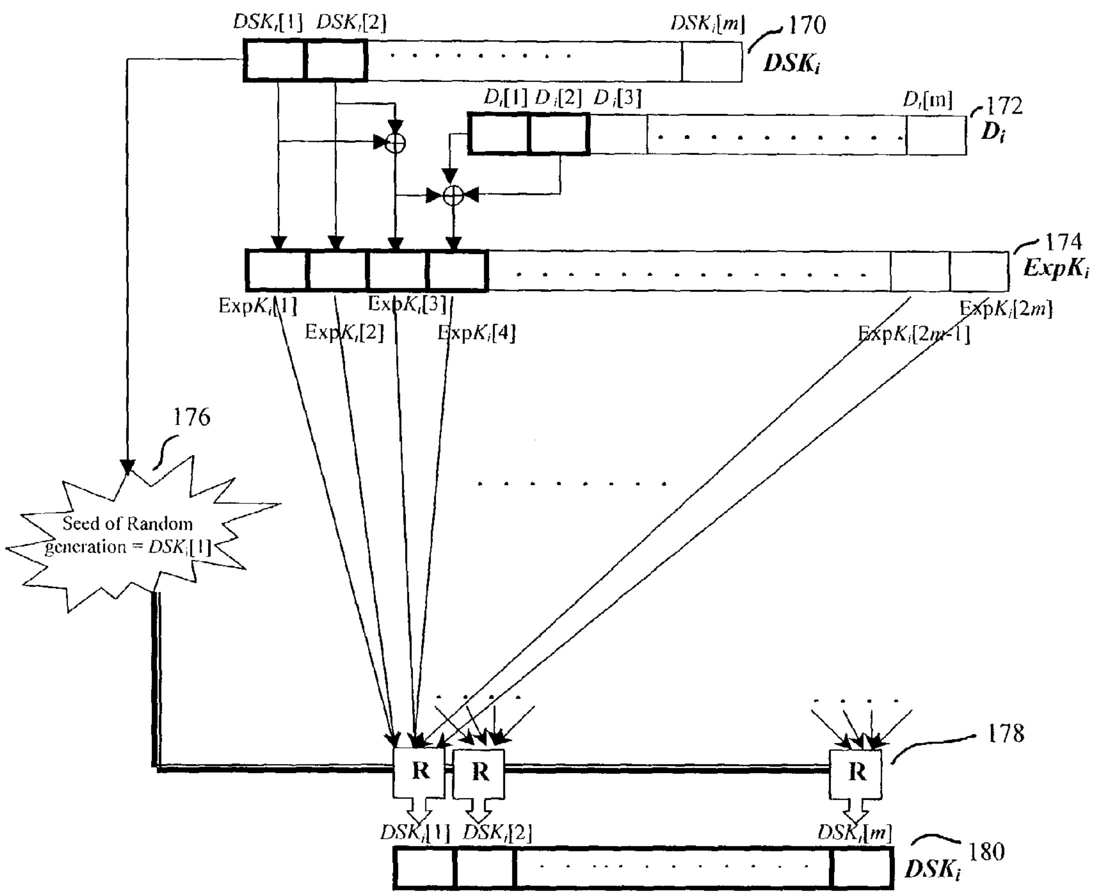

120
160
PT_i
1 $DSK_i[1]$
2 $DSK_i[2]$
m $DSK_i[m]$
C_i
$C_i[1]$ $C_i[2]$ $C_i[3]$ $C_i[m]$
158
162
Dynamic Byte Permutation:
Permute $C_i[j]$ with $C[PT_i[j]]$, for $j=m$ down to 1.
$C_i^{tmp}[1]$ $C_i^{tmp}[2]$ $C_i^{tmp}[3]$ $C_i^{tmp}[m]$
C_i^{tmp}
164
$DSK_i[1]$ $DSK_i[2]$ $DSK_i[3]$ $DSK_i[m]$
DSK_i
166
$D_i[1]$ $D_i[2]$ $D_i[3]$ $D_i[m]$
D_i
168

FIG. 12



DSKi[1] DSKi[2]
DSKi[m]
170
DSKi
Di[1] Di[2] Di[3]
Di[m]
172
Di
174
ExpKi
ExpKi[1]
ExpKi[2]
ExpKi[3]
ExpKi[4]
ExpKi[2m-1]
ExpKi[2m]
176
Seed of Random generation = DSKi[1]
178
R
R
R
DSKi[1] DSKi[2]
DSKi[m]
180
DSKi

FIG. 13

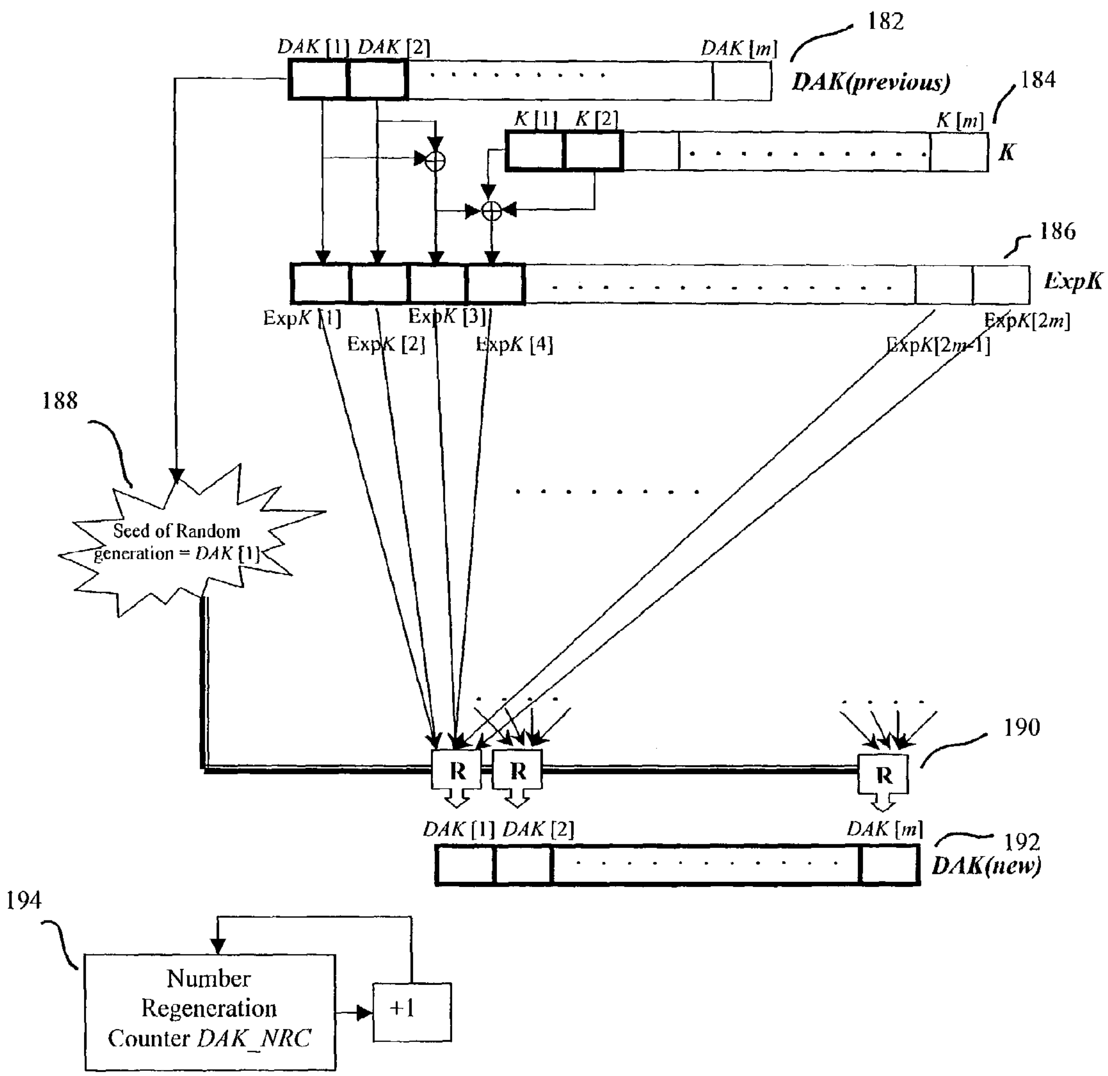
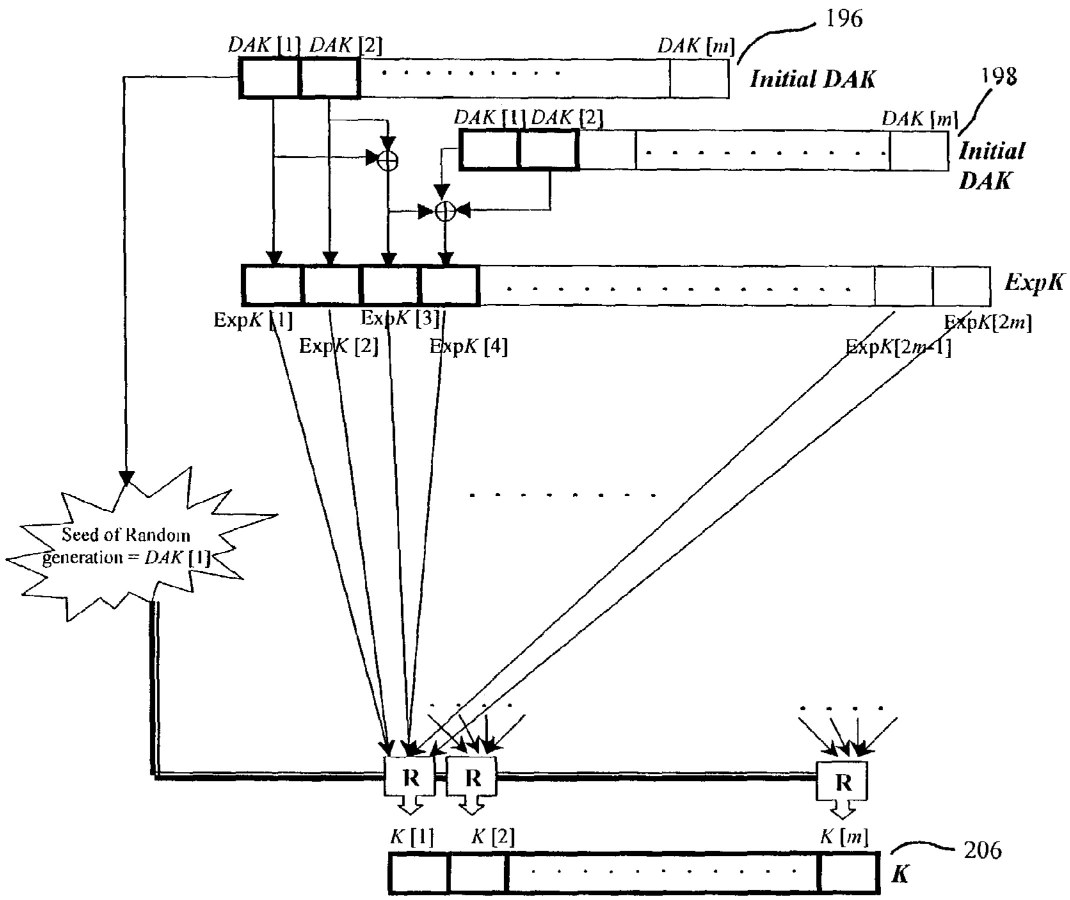
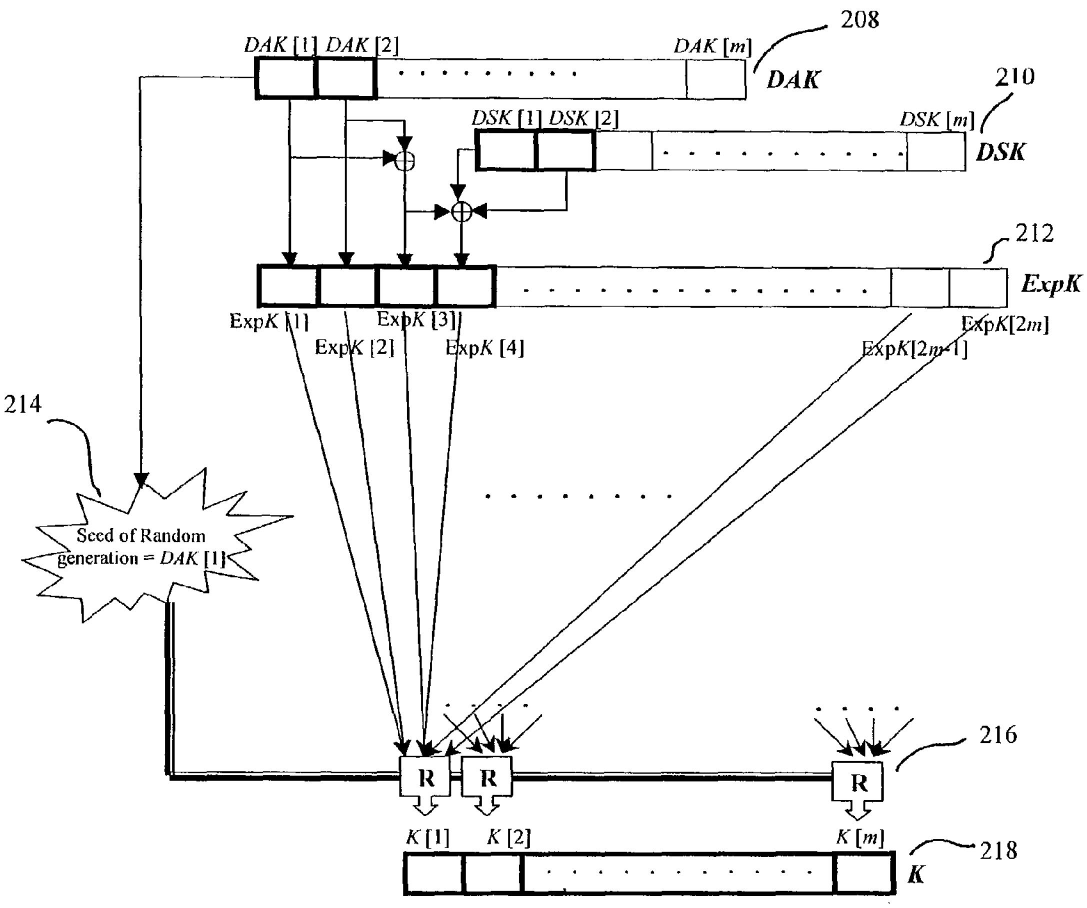

182
DAK [1] DAK [2]
DAK [m]
DAK(previous)
184
K [1] K [2]
K [m]
K
186
ExpK
ExpK [1]
ExpK [2]
ExpK [3]
ExpK [4]
ExpK[2m-1]
ExpK[2m]
188
Seed of Random generation = DAK [1]
R
R
R
190
DAK [1] DAK [2]
DAK [m]
192
DAK(new)
194
Number Regeneration Counter DAK_NRC
+1

FIG. 14



196
DAK [1] DAK [2]
DAK [m]
Initial DAK
198
DAK [1] DAK [2]
DAK [m]
Initial DAK
ExpK
ExpK [1]
ExpK [2]
ExpK [3]
ExpK [4]
ExpK[2m-1]
ExpK[2m]
Seed of Random generation = DAK [1]
R
R
R
K [1]
K [2]
K [m]
206
K

FIG. 15a



208
DAK [1]
DAK [2]
DAK [m]
DAK
210
DSK [1]
DSK [2]
DSK [m]
DSK
212
ExpK
ExpK [1]
ExpK [2]
ExpK [3]
ExpK [4]
ExpK[2m-1]
ExpK[2m]
214
Seed of Random generation = DAK [1]
216
R
R
R
K [1]
K [2]
K [m]
218
K

FIG. 15b

COMPUTER SYSTEM SECURITY VIA DYNAMIC ENCRYPTION

BACKGROUND OF THE INVENTION

1. Field of the Invention (Technical Field)

The present invention relates to the field of computer system security, more particularly to a dynamic data encryption and node authentication method and system that distributes the complexity of the encryption algorithm over the dynamics of data exchange and involves full synchronization of encryption key regeneration at system nodes, independent of the node clocks.

2. Background Art

The fundamental objective of cryptography is to enable users to communicate securely via an insecure shared data communication channel or system environment, maintaining data integrity, privacy, and user authentication. Over the past century, various cryptography systems have been developed which require a great deal of time to break even with large computational power. However, if an intruder obtains the encryption key, the encryption mechanism, and probably the entire system security, is compromised and a new key is required.

In order to make an encryption system nearly impenetrable to an intruder, two strategies are commonly used: 1) a long encryption key, and/or 2) a complex encryption function. A key of length n bits has a 2^n search space. Therefore, for large values of n an intruder needs to spend more than a lifetime to break the cipher. Also, simpler encryption functions provide a less secure encryption system. For instance, an encryption code that applies the logic XOR function is easy to decipher no matter how long the key length is. This is because the XOR operation is performed on one bit of data and its corresponding bit from the encryption key, one bit at a time. The deciphering approach of such simple encryption functions by an intruder is based on the divide-and-conquer mechanism. The intruder first deciphers individual key fragments, which is relatively uncomplicated to accomplish due to the simple linearity of the XOR function, then reconstructs the entire key once all of the individual fragments are obtained. It is more difficult to apply such a divide-and-conquer approach to break the key of a nonlinear exponential encryption function, such as used in the Rivest-Shamir-Adelman (RSA) system.

At present, there are two major cryptography system philosophies: 1) symmetric systems (static or semi-dynamic key), and 2) public key systems (static key). In symmetric systems, e.g., DES, AES, etc., a key is exchanged between the users, the sender and receiver, and is used to encrypt and decrypt the data. There are three major problems with symmetric systems. First, exchanging the key between users introduces a security loophole. In order to alleviate such a problem, the exchanged key is encrypted via a secure public key cryptography system. Second, the use of only one static encryption key makes it easier for an intruder to have an ample amount of time to break the key. This issue is addressed by the use of multiple session keys that are exchanged periodically. Third, and more importantly is the susceptibility to an "insider" attack on the key. This is referred to as the "super user" spying on the "setting duck" static key inside the system, where the time window between exchanging keys might be long enough for a super user, who has a super user privilege, to break in and steal the key.

In the RSA public key cryptography system, a user (U) generates two related keys, one is revealed to the public, deemed the "public" key, to be used to encrypt any data to be sent to U. The second key is private to U, called the "private" key, and is used to decrypt any data received at U, which was encrypted with the corresponding public key. The RSA cryptography system generates large random primes and multiplies them to get the public key. It also uses a complex encryption function such as mod and exponential operations. As a result, this technique is unbreakable in the lifetime of a human being for large keys, e.g., higher than 256 bits, and also eliminates the problem of the insecure exchange of symmetric keys, as in a DES system. However, the huge computational time required by RSA encryption and decryption, in addition to the time required to generate the keys, is not appealing to the Internet user community. Thus, RSA cryptography is mainly used as "one shot" solid protection of the symmetric cryptography key exchange.

In the RSA public key system, if a first user (U_A) requests a secure communication with a second user (U_B), the latter will generate a pair of encryption keys: public E_B and private D_B. An internal super user spy (S), with a helper (H) intruding on the communication line externally, can easily generate its own pair of keys, a public E_S and private D_S, and pass D_S and E_B to H. Then S can replace the public key E_B with its own public key E_S. Thus, all data moving from U_A to U_B will be encrypted using E_S instead of E_B. Now H can decrypt the cipher text moving between U_A and U_B using the private key D_S, store it, and re-encrypt it using the original E_B, in order for U_B to receive and decrypt it without any knowledge of the break that occurred in the middle. Such an attack is typically called the "super-user-in-the-middle" attack.

Even though they are secure against outsider attack, both the symmetric and public key cryptography systems are still vulnerable to insider attacks. By obtaining the key at any time of a secure session, an intruder can decipher the entire exchanged data set, past and future. Further, a super user can easily steal a static symmetric key and send it to an outside intruder to sniff and decrypt the cipher text, particularly in the DES and AES systems.

A common way to protect a static encryption key is to save it under a file with restricted access. This restriction is not enough, however, to prevent a person with super-user privilege from accessing the static key in the host file. Even when keys are changed for each communication session, for example in the Diffie-Hufman system, there is a time window enough for the super-user to obtain the semi-static key. In most crypto systems, once the key is found the previous and future communicated data are no longer secure.

Various other attempts have been made to circumvent intrusion by outside users through encryption of communicated data. Examples of such methods include that described in U.S. Pat. No. 6,105,133 to Fielder, et al., entitled, "Bilateral Authentication and Encryption System;" U.S. Pat. No. 6,049,612 also to Fielder, et al., entitled, "File Encryption Method and System;" and U.S. Pat. No. 6,070,198 to Krause, et al., entitled, "Encryption with a Streams-Based Protocol Stack." While the techniques described in these patents may be useful in preventing unwanted intrusion by outsiders, they are still prone to attack by the super-user-in-the-middle.

The present invention alleviates the problems encountered in the prior art, providing continuous encryption key modification, one key for each data record. New keys are generated from the previous key and data record, and are used to encrypt the subsequent data record. The key lifetime is equal to the time span of record encryption, which is too small for an intruder to break and a super-user to copy. The present invention also reduces computational overhead by breaking the complexity of the encryption function and shifting it over the dynamics of data exchange. Speed is also improved through the use of a simple XOR logic encryption function. A shuf-

fling mechanism based on a dynamic permutation table, which is generated from the current session key, is coupled with the XOR logic operation, to strengthen the encryption function. Encryption is fully automated and all parties, the source user, destination user, and central authority, are clock-free synchronized, and securely authenticated, at all times.

The present invention also alleviates the "super-user-in-the-middle" attack. An intruder must obtain an entire set of keys, at the right moment, without being noticed, in order to decrypt the entire ciphered message. The dynamic key encryption system of the present invention is deployable at any level of a system, as there is complete synchronization between parties.

SUMMARY OF THE INVENTION (DISCLOSURE OF THE INVENTION)

The present invention is a method of providing a secure data stream between system nodes. The method includes creating data at a source user node and regenerating a new encryption key, or dynamic session key (DSK), at the node using the created data and a previous DSK. The new DSK is generated by performing a logic operation on a previous DSK and a data record. Preferably, the logic operation is an XOR operation. The data record and previous DSK are XORed to form an expanded key, ExpK. Bytes are randomly selected from the ExpK to generate the new DSK, using a byte from a previous DSK as a seed of random generation.

A data record is encrypted with a DSK by performing a logic XOR operation on the data and DSK to form a temporary cipher. Portions of the cipher are then permuted to form another cipher, which is then transmitted over a data stream to the destination user node.

A block of n data records at a time, each record of m bytes, and a corresponding number of n DSKs, each of m bytes, are combined to form n new DSKs. The n new DSKs are then used to encrypt the subsequent block of n data records. The process continues until all data records are encrypted and transmitted from the source user node.

The method further comprises the step of receiving encrypted data at a destination user node and decrypting the received encrypted data with a DSK. Once the encrypted data is decrypted, new DSKs are regenerated at the destination user node using the decrypted data and a previous DSK. A central authority node is used to assure that both the source and destination nodes begin with the same initial DSK so that the destination node can properly decrypt the received encrypted data.

The present invention is further a method of authenticating one system node to another system node comprising the steps of: generating an authentication key at a central authority node; transmitting the authentication key to a user node; and starting a daemon at the central authority node and a daemon at the user node. These daemons regenerate new dynamic authentication keys (DAKs) every δt and maintain a corresponding number-regeneration-counter at each node.

A new DAK is generated by performing an XOR logic operation on a previous DAK and an auxiliary key, K. Performing the XOR operation on the previous DAK and auxiliary key, K forms an expanded key, ExpK. Bytes are randomly selected from this expanded key to generate the new DAK, where a byte from a previous DAK is used as a seed of random generation. The auxiliary key, K, is formed by performing an XOR logic operation on the DAK and last used DSK. Performing the XOR operation on the previous DAK and DSK forms an expanded key, ExpK. Bytes are randomly selected from this expanded key to generate the auxiliary static key, K, where a byte from the DAK is used as a seed of random generation. If no previous DSK exists, because it is the user's initial session, then K is formed from the initial DAK, only, where a copy of the DAK replaces the nonexistent DSK.

The method of the present invention also includes synchronizing one node to another node. Synchronization is achieved by aligning the DAKs of one node to another and comprises the steps of: forking a child communication process at a user node which freezes the DAK and its number regeneration count at the user node, transmitting the frozen DAK number regeneration count from the user node to a central authority node; forking a child communication process at the central authority node which snapshots the DAK number regeneration count for that user at the central authority node; comparing the frozen and snapshot DAK number regeneration counts of the user and central authority nodes; and aligning the DAKs of the user and central authority nodes according to the comparison.

In addition to being aligned, nodes are authenticated to one another by generating a nonce, N, at the central authority node; encrypting the nonce with a frozen DAK; transmitting the nonce and the encrypted nonce to a user node; decrypting the encrypted nonce at the user node with a frozen DAK; and comparing the decrypted nonce with the nonce. Then, both nodes calculate N^2. If the user node determines that the decrypted nonce is equal to the nonce, then the user node transmits an encrypted version of N^2 back to the central authority node, encrypting it with its DAK. The central authority node receives the encrypted N^2 and decrypts it with its DAK. Finally, the central authority compares the decrypted N^2 to N^2, and if it finds them equal, mutual authentication was successful.

Once a central authority has synchronized and authenticated itself to a user, the central authority generates an initial DSK; encrypts the initial DSK with the DAK that is aligned with that user; and transmits the encrypted initial DSK to that user. The same initial DSK is sent to both the source and destination users, encrypted with their corresponding aligned DAKs so that they can start a secure communication session. After having received the identical initial DSK from the central authority, each user, source and destination, regenerates new DSKs based on previous DSKs and the data that was created at the source user node.

The present invention is further a system for providing a secure data stream between a source programmable apparatus and a destination programmable apparatus. The system comprises: a source programmable apparatus; a data stream created by the source programmable apparatus; means for encrypting data of the data stream with a DSK; and means for regenerating a new DSK using data from the data stream. The system also includes a destination programmable apparatus in electrical communication with the source programmable apparatus; means for transmitting encrypted data to the destination programmable apparatus; means for decrypting the encrypted data received at the destination programmable apparatus with a DSK; and means for regenerating a new DSK using decrypted data.

A primary object of the present invention is to provide a dynamic encryption method and system having no static keys, public or private, that are susceptible to a security breach. Another primary object of the invention is to provide improved security to a data stream between a source and destination user, in particular to provide improved security against a super-user having insider privileges. Another primary object of the present invention is to provide such improved security at a high speed, in a secure system environment, via mutually authenticated users and CAs.

A primary advantage of the present invention is that it is fully automated, with all system nodes synchronized and mutually authenticated, to ensure security. Another primary advantage of the invention is that it is simple and fast, yet secure against spying by an internal super-user or outside intruder due to the large number of dynamic keys (n) that would need to be broken or compromised—one key per ciphered data record, (n) parallel sessions of encryption, and zero entropy of the ciphered text. Yet another primary advantage of the invention is that it minimizes the exchange of keys between users and/or the CA. Still yet another advantage of the invention is that an initial DAK is securely exchanged between a user and CA which is continuously regenerated during the entire life of the user, allowing the user and CA to synchronize (realign DAKs) and authenticate to one another as needed for a communication session or when there is a disaster misalignment between a user and CA.

Other objects, advantages and novel features, and further scope of applicability of the present invention will be set forth in part in the detailed description to follow, taken in conjunction with the accompanying drawings, and in part will become apparent to those skilled in the art upon examination of the following, or may be learned by practice of the invention. The objects and advantages of the invention may be realized and attained by means of the instrumentalities and combinations particularly pointed out in the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated into and form a part of the specification, illustrate a preferred embodiment of the present invention and, together with the description, serve to explain the principles of the invention. The drawings are not to be construed as limiting the invention.

FIG. **1***a* is a diagrammatic overview of a central authority (CA) generating daemons to manage users' dynamic authentication keys (DAKs) in accordance with the present invention;

FIG. **1***b* is a diagrammatic overview of secure communication between users in accordance with the present invention;

FIG. **2** is a diagrammatic illustration of a user registration request to a CA in accordance with the present invention;

FIG. **3** is a diagrammatic overview of synchronization and authentication between users and a CA, and initial generation of the DSK by the CA in accordance with the present invention;

FIG. **4** is a diagrammatic illustration detailing the method of FIG. **3**;

FIG. **5** is a diagrammatic illustration of synchronization of DAKs between a CA and a user in accordance with the present invention;

FIG. **6***a* is a diagrammatic illustration of the method whereby a CA authenticates a user in accordance with the present invention;

FIG. **6***b* is a diagrammatic illustration of the method whereby a user authenticates a CA in accordance with the present invention;

FIG. **7***a* is a diagrammatic illustration of the method whereby a CA freezes and resumes regeneration of users' DAKs upon occurrence of a CA shutdown event, in accordance with the present invention;

FIG. **7***b* is a diagrammatic illustration of the method whereby a user freezes and resumes regeneration of its DAK upon occurrence of a user shutdown event, in accordance with the present invention;

FIG. **8** is a diagrammatic illustration of secure communication establishment at the source user node in accordance with the present invention;

FIG. **9** is a diagrammatic illustration of secure communication establishment at the destination user node in accordance with the present invention;

FIG. **10** is a diagrammatic illustration of a user handshaking method with a CA in accordance with the present invention;

FIG. **11** is a diagrammatic illustration of the dynamic encryption and permutation method of the present invention;

FIG. **12** is a diagrammatic illustration of the dynamic decryption and permutation method of the present invention;

FIG. **13** is a diagrammatic illustration of the DSK regeneration method of the present invention;

FIG. **14** is a diagrammatic illustration of the DAK regeneration method of the present invention;

FIG. **15***a* is a diagrammatic illustration of the formation of the auxiliary static key using the initial DAK in accordance with the present invention; and

FIG. **15***b* is a diagrammatic illustration of the formation of the auxiliary static key using the DAK and previous DSK in accordance with the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Best Modes for Carrying Out the Invention

The present invention is a dynamic symmetric key encryption method and system for network security. The invention is implemented between end-users (U) and central authentication authorities (CAs) for user authentication purposes, and between end-users for secure exchange of digital data. The users and CA reside at their respective nodes, or programmable apparatuses, such as at one or more computers, and are in electrical communication, such as via a computer network. Communicated data flows over a data stream between the user and CA programmable apparatuses. Computer-readable memory provides storage for the data, dynamically changing keys, and other variables, as needed to allow for the regeneration of subsequent dynamic keys, and to carry out other necessary processes within the computer. Means are provided on the programmable apparatuses for performing all of the various methods involved in the dynamic encryption method. Such means include primarily computer-readable means, such as software, and the necessary related hardware.

The encryption method of the present invention distributes the complexity of the encryption algorithm over the dynamics of the data exchange, involving the exchanged data as well as the previous key in the process of regenerating a "dynamic" encryption key; i.e., $KEY_{new}=F(KEY_{previous}, DATA\_RECORD_{current})$, where F is a key regeneration function. Thus, there is a newly generated key for the encryption of every data record, yielding zero entropy between the cipher and the plain data. There are no static keys, public or private, that are susceptible to a security breach. In order to guarantee security, the encryption method of the present invention is preferably deployed in a system of registered end-users with a CA, whereby the CA maintains user connections' authentication and secures distribution of symmetric encryption session keys.

The invention employs two types of dynamic keys, namely dynamic authentication keys (DAK) and dynamic session keys (DSK). The former is used to mutually authenticate users and CAs; it is continuously regenerated throughout the existence of the user and the CA. The latter exists when the need arises for a secure data exchange session between users; its regeneration is maintained only through the life cycle of such a session. The placement of the initial DSK at users' nodes is securely carried out by the CA, and encrypted using the users' DAKs. The CA maintains an array of DAKs, one per user.

The invention further employs an auxiliary static key K, which is formed based on the DSK and the DAK, given that the user has established a session; otherwise, it is formed from the initial DAK only. This static key is continuously involved in the regeneration of the DAK. The auxiliary static key adds another dimension of security to the process against insider attacks, as it involves more dynamics to the regeneration of the DAK by allowing the contribution of the DSK to the process, every new communication session between users. The static nature of K is not to be exploited by the attacker since its exploitation does not lead to any important information; its relation to the DSK and the DAK is not reversible, i.e., K is manufactured from DSK and DAK, yet neither DSK nor DAK can be obtained from K.

The major role of the CA is to maintain the registered users' DAKs generation and the secure delivery of symmetric session keys (DSKs) between the users. The CA also authenticates the source user to the destination user and vice versa, while authenticating itself to both users. Unless a user is pre-registered with the CA, it is nearly impossible for an intruder to have the same synchronized dynamic key with the CA, as a registered user. The only explicit user identity verification is carried out once, at the time the user first registers with the CA. Any consequent authentication is implicitly processed, without the need to exchange any plain keys, which would require a secure channel. The authentication method involves the three parties to any connection: source user, destination user, and CA, authenticating each other via their corresponding synchronized dynamic key.

Referring to FIG. **1***a*, a diagrammatic overview of a CA generating daemons to manage users' dynamic authentication keys (DAKs) in accordance with the present invention is shown. Registration of trusted users, users that have for example, provided a valid certificate of authentication or who are referred by a previously-registered third party user, occurs over a secure channel, for example, using RSA encryption, where the CA provides the user with an initial DAK. Referring to FIG. **1***b*, a diagrammatic overview of user communication encrypted with a dynamic session key (DSK) initially generated and sent by a CA is shown. Upon any user's connection request, a communication child process is forked at the CA, as well as at the user node. Each child process runs on the behalf of its parent during the whole communication session in order to avoid disturbance of the DAK generation process in the event of synchronization or authentication failure, for example, due to a false connection request, hardware failure, etc.

The CA is responsible for secure user authentication and generation of the initial trusted user dynamic authentication key, DAK. Every user must register with the CA in order to obtain its initial DAK, which is exchanged via a secure channel, e.g., using RSA. Upon the initiation of any new user's secure data exchange session, both end-users and the CA freeze their DAK generation and synchronize by establishing the same dynamic change. After the user and the CA are synchronized (their DAKs are aligned), they both yield the same randomly generated DAK at both nodes. This DAK is used in the authentication method as the encryption and decryption key, because it is maintained only by the CA and the user.

Referring to FIG. **2**, a diagram further illustrates the user registration method with a CA. The user starts the registration process by sending a request **10**, effectively requesting the generation of an initial value of its DAK, to the CA including authenticated documentation of its identity and purpose of registration, i.e. revealing it is a trusted user. Upon approval of the user's credentials by the CA, the CA starts a daemon related to the requesting user, and randomly selects an initial DAK, sending a copy to the user via a secure channel **12**, for example, using the well-known RSA technique where the CA uses the user's public key to encrypt the newly generated DAK. Then the CA starts a daemon that permanently regenerates the DAK. Upon the reception of the initial DAK, the user starts a daemon in order to permanently regenerate the DAK. From that point forward, the user and CA randomly regenerate the next DAK every δt period, **14**, **15**, based on the previous generated DAK and the auxiliary static key K. The user and CA also maintain a number-regeneration-counter (NRC). This method of regenerating new DAKs is depicted in greater detail in FIGS. **14** and **15**.

Referring to FIG. **14**, a diagram illustrates the continuous DAK regeneration method, where DAK[j] represents the j^{th} byte of the dynamic authentication key, K[j] represents the j^{th} byte of the auxiliary static key, ExpK[h] represents the h^{th} byte of an "expanded key", $1 \leqq j \leqq m$, $1 \leqq h \leqq 2m$. Each unit R represents the random selection of one byte among the ExpK's 2m bytes. An expanded key, ExpK **186** of twice the size of the DAK **182** is generated as follows. Each of the DAK, K, and ExpK are divided into (m/2) regions, indexed from 1 to (m/2). Each region of DAK **182** and DSK, **184** is of length 2 bytes, and each region of ExpK, **186** is 4 bytes. The indices of the four consecutive bytes of any region r in the ExpK are indexed with the values: 4r−3, 4r−2, 4r−1, and 4r The indices of the two consecutive bytes of any region r, in the DAK and K, are indexed with the values: 2r−1 and 2r. The four bytes of region r in the ExpK are filled from DAK and K as follows:

$$ExpK[4r-3] \leftarrow DAK[2r-1]$$

$$ExpK[4r-2] \leftarrow DAK[2r]$$

$$ExpK[4r-1] \leftarrow (DAK[2r-1])XOR(DAK[2r])$$

$$ExpK[4r] \leftarrow (DAK[2r-1])XOR(DAK[2r])XOR(K[2r-1])XOR(K[2r])$$

It will be understood by those of skill in the art that, ExpK can alternatively be comprised of any number of bytes, e.g., 2m, 3m, 8m, etc., and the invention is not limited to any particular size for ExpK. If ExpK were of a different, greater number of bytes than 2m, then the logic operation would be altered as needed to fill the bytes of ExpK. It will also be understood that the byte positions chosen to be XORed together to fill the positions of ExpK could similarly be altered, and still remain within the inventive scope of the dynamic encryption and authentication method. Similarly, alternative logic operations could be substituted for the XOR operation.

Once ExpK is created, a random selection of m bytes from the 2m bytes of ExpK is taken **190** based on the first byte of the DAK (DAK[1]) as the random function seed **188**. Alternatively, a randomly selected byte of the DAK can be used as the random function seed. This function generates a sequence of m random numbers in the range between 1 and 2m each of

which represents the index of the ExpK byte to be placed as the next byte of the regenerated DAK **192**. The operation depicted in FIG. **14** is performed at both the user and the CA nodes. In order to maintain synchronization control, a number-regeneration-count for the dynamic authentication key (DAK_NRC) **194** is maintained and incremented after each DAK regeneration. This method can be performed periodically or aperiodically (with mutual consent of CA and the user) according to the implementation of the invention.

Continuing on to FIGS. **15***a* and **15***b*, a diagram illustrates the method of forming the auxiliary static key K. FIG. **15***b* illustrates the formation of K using the CA-user aligned DAK, where DAK[j] represents the j^{th} byte of the dynamic authentication key, DSK[j] represents the j^{th} byte of the last dynamic session key, K[j] represents the j^{th} byte of the auxiliary static key K, and ExpK[h] represents the h^{th} byte of an "expanded key", $1 \leqq j \leqq m$, $1 \leqq h \leqq 2m$. Each unit R represents the random selection of one byte among the ExpK's 2m bytes. Each static key K is created using the CA-user aligned DAK, **208** and the last user session DSK, **210**. First, an expanded key (ExpK), **212**, of twice the size of K is generated. Each of the DAK and the DSK is divided into (m/2) regions, indexed from 1 to (m/2). Each region of the DAK and the DSK is of length 2 bytes, and each region of ExpK is 4 bytes. The indexes of the four consecutive bytes of any region r, in the ExpK, are indexed with the values: 4r−3, 4r−2, 4r−1, and 4r. The indexes of the two consecutive bytes of any region r, in the DAK and the DSK, are indexed with the values: 2r−1 and 2r. The four bytes of region r in the ExpK are filled from DAK and DSK as follows:

$$\mathrm{Exp}K[4r-3] \leftarrow \mathrm{DAK}[2r-1]$$

$$\mathrm{Exp}K[4r-2] \leftarrow \mathrm{DAK}[2r]$$

$$\mathrm{Exp}K[4r-1] \leftarrow (\mathrm{DAK}[2r-1])\mathrm{XOR}(\mathrm{DAK}[2r])$$

$$\mathrm{Exp}K[4r] \leftarrow (\mathrm{DAK}[2r-1])\mathrm{XOR}(\mathrm{DAK}[2r])\mathrm{XOR}(\mathrm{DSK}[2r-1])\mathrm{XOR}(\mathrm{DSK}[2r])$$

Then, a random selection of m bytes from the 2m bytes of ExpK is performed, **216**, based on the DAK[1] byte as the random function seed, **214**. Alternatively, any randomly selected byte can serve as the random function seed. This function generates a sequence of m random numbers, in the range between 1 and 2m, each of which represents the index of the byte to be placed as the next byte of K, **218**. This operation is performed at both the source and destination user nodes. This method is used to form K when a user communication session is taking place and DSKs are being regenerated.

FIG. **15***a* illustrates the formation of K, **206** using the two copies of the initial CA-user aligned DAK, **196**, **198**. This method is used to form K when a communication session is not taking place, i.e., the user DSK does not exist. Thus, the mechanism of FIG. **15***a* follows the same pattern as FIG. **15***b*, except replacing the non-existing DSK by another copy of the initial DAK.

Returning to FIG. **3**, a diagrammatic overview of synchronization and authentication between users and a CA, and initial generation of the DSK by the CA is shown. DAK regeneration starts at both the user and the CA nodes, upon user registration, where each generates the same sequence of random DAKs, with the initial DAK as a seed, even after the user-CA connection is terminated. Thus, key exchange between users, as well as between users and the CA, is minimized to maintain a high level of security. Users and the CA instead remain synchronized at all times with respect to DAK regeneration. When the user-CA connection is terminated after initial request for user registration with the CA, synchronized DAK regeneration continues off-line. Permanent regeneration of the DAK is maintained via a daemon running at each registered user, and a corresponding daemon running at the CA.

With continuing reference to the center portion of the diagram of FIG. **3**, in order to establish a connection between a source user U_s and a destination user U_d, U_s's DAK regeneration daemon forks a child communication process U_s_COM, which freezes its version of the DAK and its NRC, and sends a connection request to the CA including synchronization information. Upon the reception of such request, the CA's communication server forks a communication child process CA_COM, which will notify U_d of the connection with U_s. Upon the acceptance by U_d to the connection, U_d's DAK regeneration daemon forks a child communication process U_d_COM, which freezes its version of the DAK and its NRC, and sends synchronization information back to the CA_COM. Then, the CA_COM snapshots both users' DAKs/NRCs from their corresponding CA's DAK daemons, and starts the synchronization and authentication processes with both users' communication child processes.

Upon successful mutual synchronization and authentication involving the three session parties, the CA_COM generates a random dynamic session key (DSK) and encrypts it using the aligned DAK of each user, and sends it to both users. After receiving the encrypted DSK, each of the two users' child communication process decrypts it using its respective aligned DAK, and starts a secure communication session with the other child communication process, via the decrypted DSK.

In case of failure, the child processes are terminated without interruption to the parent processes and/or daemons. This feature provides protection against a "synchronization disturbance" attack. In such an attack, an intruder imitating a registered user or a CA might force the three DAK number-regeneration-counters (NRCs) to freeze, which will create a chaotic state, but only in the child processes. The counters are continuously counting, i.e., DAKs are continuously regenerated, in the daemons without stopping, except when rebooting. The child processes snapshot, or "freeze", the DAK/NRC for use in the synchronization, authentication, and secure DSK exchange. Thus, the continuously running DAK daemons are unaffected in the parent processes in the event of a failure to establish a connection, or in the event of a synchronization disturbance.

However, in the event of successful synchronization and authentication, the child processes at the users' nodes return the aligned DAK and the newly generated DSK to their respective parent daemons in order to initialize a new state for DAK regeneration, forcing the daemon to discard the current value of DAK and DSK (if exists), and consider the newly returned versions in the DAK regeneration process. Also, the CA_COM return the two aligned DAKs and the newly generated DSK to their respective DAK daemons at the CA in order to initialize a new state for DAK regeneration, forcing both local users' daemons to discard their current DAK and DSK (if exists) values, and consider the newly returned versions in the DAK regeneration process. Then, the CA_COM terminates successfully, whereas the U_s_COM and U_d_COM start a secure communication.

Referring to FIG. **4**, a diagrammatic overview of synchronization, authentication, and generation of DSK by a CA in response to a request from a source user (U_s) to communicate with a destination user (U_d) is provided. (See also FIG. **3**.) Source user U_s requests a DSK generation from CA to communicate with destination user U_d, and sends its frozen

DAK_NRC along with the request, **16**. The CA forks a CA_COM, which snapshots the two users DAKs, namely CA_DAK[U_s] and CA_DAK[U_d], and requests U_d to send its DAK_NRC, **18**. This request notifies U_d that U_s is trying to establish a secure communication with it. Once the CA has received the U_d DAK_NRC, **20**, the synchronization process is initiated, **22**.

Synchronization ensures that the CA has DAK values identical to the users' DAK values, despite message propagation delay. The CA ensures that its locally snapshot DAK for U_s and the corresponding frozen DAK at U_s's node are aligned, and also ensures that its locally snapshot DAK for U_d and the corresponding frozen DAK at U_d's node, are aligned. To compensate for propagation delay and align the corresponding CA's DAK with each of the parties' DAKs, the difference (x) in the number of key regeneration counts (NRCs), between CA and each user, will be considered by the lagging party, which will regenerate its DAK an extra x times (See FIG. **5**.)

If alignment is not achieved with both users, **26**, CA ignores the synchronization effects from the non-synchronized user(s), sends an abort message to both users before killing its communication child process, and cancels the communication due to lack of synchronization.

In the event of successful synchronization with both users, **24**, the CA launches an authentication method, **28**, to certify their identity. (FIGS. **6***a* and **6***b*.) If successful authentication of both users is not achieved, **32**, CA ignores any synchronization effects of the non-authenticated user(s), sends an abort message to both users before killing its communication child process, and cancels the communication due to lack of authentication. If both users are fully authenticated and synchronized with CA, **30**, the CA randomly generates an initial DSK and sends it to both users, encrypted with each user's corresponding aligned DAK, to begin data exchange, **34**. This encryption process, **34**, is identical to the process described in FIG. **11**, except that DSK_i is replaced by DAK, and D_i is replaced by the initial DSK. After the entire process is completed, successful or unsuccessful, the CA_COM terminates and returns its status to the parent process.

Referring to FIG. **5**, a diagram illustrates synchronization of DAKs between a CA and a user, based on the number-regeneration-count for the DAK at each node. Initially, the number-regeneration-count of the DAK for user (U) at the CA, (CA_DAK_NRC[U]), is compared to the number-regeneration-count of the DAK at the user's node (U_DAK_NRC), **36**. If the two NRCs are equal, then the CA and user are synchronized. If the comparison of the NRCs is outside of a predetermined acceptable range, a "failure-to-synchronize" message is reported, **40**. This occurs when |CA_DAK_NRC [U]−U_DAK_NRC|, **38**, is larger than a predetermined value, and the communication is terminated.

If the comparison of the NRCs is within the predetermined acceptable range, then the lagging party performs a number of DAK regenerations equal to the calculated difference in order to synchronize (i.e., align the DAKs) with the other party. For example, if the CA NRC lags behind that of the user, **42**, then the CA performs (U_DAK_NRC−CA_DAK_NRC[U]) regenerations of its DAK in order to align with that of U, **44**. If the user NRC lags behind that of the CA, the CA sends a "synchronize" message, including the calculated NRC difference, **46**, so that the user can perform the appropriate number of regenerations to synchronize with the CA. Once the user performs the regenerations, it signifies that it has done so to the CA, **48**.

Once the parties are synchronized, mutual authentication of DAKs is performed to ensure the parties indeed share the same DAK. FIGS. **6***a* and **6***b* illustrate the mutual authentication method. FIG. **6***a* illustrates authentication of a user by a CA, and FIG. **6***b* illustrates authentication of a CA by a user. The process begins by CA generating a random number, or nonce, N. CA sends N and E(N) to the user, where E(N) is the encrypted version of N using the CA_DAK[U], the shared and aligned DAK, as well as an "authenticate" message, **50**. The user decrypts E(N) using its frozen DAK, which should be identical to CA_DAK[U], that of the CA, **64** and verifies that D(E(N))=N, **66**. The user thus authenticates the CA, **68**. If D(E(N)) does not equal N, then the user failed to authenticate the CA and the connection is aborted, **70**.

When the user has successfully authenticated the CA, **68**, the user encrypts N^2 with its DAK, ($E(N^2)$), and sends $E(N^2)$ back to the CA, as part of the authentication acknowledgment to complete the authentication process, **68**. Upon receiving the authentication acknowledgment back from the user, **52**, CA decrypts the received ciphered number, again using the aligned CA_DAK [U], **54** and compares the decrypted value with the value of N^2, **56**. If they are not equal, the CA reports a failure to authenticate the user, **60**, and aborts the establishment of a connection. If they are equal, the user is successfully authenticated by the CA, i.e., has been registered with the CA, **58**. CA performs the same mutual authentication method with the second user before user-to-user communication takes place. When all parties have been mutually authenticated, CA proceeds to DSK generation, **62**, the last step of FIG. **4**.

As systems are prone to unpredicted shutdown events, such as power loss, the dynamic encryption method and system automatically freezes and resumes key regeneration upon occurrence of such events. Referring to FIG. **7***a*, a diagram illustrating freezing and resuming regeneration of DAKs when a CA experiences a shutdown event, is shown. FIG. **7***b* shows freezing and resuming regeneration of DAKs when a user experiences a shutdown event. Referring to FIG. **7***a*, a CA is shown to experience a shutdown event, **72**. The CA immediately sends a "freeze-DAK-regenerating" message to all previously registered users, **74**, as part of its shutdown handler routine. Meanwhile, the CA saves all users' DAKs into a temporary file, **76**. After shutting down, **78**, for a time period τ, the CA reboots and reloads all the previously saved DAKs. The CA then requests the DAK_NRC from all users to ensure validity of the current DAKs, **80**. Then, the CA starts the synchronization process, **82** as described with respect to FIG. **5**. The CA then initiates the mutual authentication process with all successfully synchronized users, **84**. (FIGS. **6***a* and **6***b*.) Finally, the CA sends a "resume-DAK-regenerating" message to its registered and successfully synchronized and authenticated users, in order to resume realigned regeneration of the DAKs, **85**. A similar method is performed by the user in the event of a user's system shutdown, as depicted in FIG. **7***b*.

After a user is registered with a CA, the user may request a secure communication with another user. Referring to FIG. **8**, a diagram illustrates secure communication establishment at the source user (U_s) side. The source user first determines whether it is registered to the CA, **86**. If not, the user performs the registration procedure and receives its initial DAK via a common secure channel, **88**. (See also FIG. **2**.) Then, the source user's DAK daemon forks a child communication process U_s_COM, **90**, which freezes its DAK generation and runs on behalf of the user until the end of the users' session communication. U_s_COM requests a secure connection establishment with the destination user (U_d) from the CA, and sends its frozen DAK_NRC for synchronization purposes. In

the event of successful handshaking with the CA, **92** (FIG. **10**), the source user receives an initial dynamic session key DSK, from the CA, **94**.

The U_s-U_d data exchange session uses the shared symmetric DSK sent by the CA to both users. However, to increase the level of dynamic encryption to n crypto parallel streams, a set of n updated DSKs is derived randomly from the initial DSK, used as a seed, sent by CA. The source user message is stepped through n records, or a "block", at a time, and the first n DSKs generated are used to encrypt the first block of n records. After that, n DSKs are regenerated for each consecutive block, as a function of the previously generated n DSKs and their corresponding n data records. (See FIG. **13**.)

This process is depicted in FIG. **8** after the source user has had successful handshaking, **94**. The source user first generates randomly n different DSKs (DSK_i, where $1 \leqq i \leqq n$), of the same size as the initial DSK, **96**. Then, n data records of the same size as the size of DSK ($Record_i$, where $1 \leqq i \leqq n$) are extracted from the input data, **98**, and encrypted **100**. Next, for every $Record_i$ and its current DSK_i as input, a new DSK_i is regenerated, **102**. This is depicted in greater detail in FIG. **13**, on a byte-by-byte basis. Finally, the n ciphered records are transmitted to U_d, **104**. The encryption method described at **100** of FIG. **8** is illustrated in greater detail in FIG. **11**.

Turning to FIG. **11**, a diagram illustrates the encryption method described at **100** of FIG. **8**. In FIG. **11**: $D_i[j]$ is the j^{th} byte of the i^{th} data record, $DSK_i[j]$ is the j^{th} byte of the corresponding i^{th} DSK, $C_i^{tmp}[j]$ is the j^{th} byte of the i^{th} "temporary cipher record," $C_i[j]$ is the j^{th} byte of the produced i^{th} cipher record, and $PT_i[j]$ is the j^{th} byte of the i^{th} permutation table, $1 \leqq i \leqq n$ and $1 \leqq j \leqq m$, for n records, each of which is of m bytes.

The data record D_i and the DSK_i are used to produce C_i^{tmp}, which is permuted based on PT_i to produce cipher record C_i, $1 \leqq i \leqq n$, producing n cipher records. The encryption method uses a simple XOR logic operation as an encryption function between a DSK_i and its corresponding data record D_i. The data byte $D_i[j]$, **146**, is XORed with the corresponding key byte $DSK_i[j]$, **148**, resulting in a temporary cipher byte $C_i^{tmp}[j]$, **150**. To shuffle the C_i^{tmp} bytes around, a permutation table PT_i, **152** is generated as a copy of the corresponding DSK_i. The PT_i bytes are scanned from index 1 to index m, using each entry's index and its associated entry value, as indices of two bytes in the temporary cipher record to be swapped. The permutation is performed as follows: permute $C_i^{tmp}[j]$ with $C_i^{tmp}[PT_i[j]]$, for $1 \leqq j \leqq m$, which results in the final cipher record C_i, **156**. After n data records are encrypted with n DSKs as aforementioned, the final cipher block, made up of n cipher records (C_i, where $1 \leqq i \leqq n$), is available for transmission, and a set of n new DSKs is regenerated for encryption of the next data block, as shown in FIG. **13**.

Returning to FIG. **8**, the final cipher block is transmitted to U_d, **104** after having generated the n updated DSKs, **102**, for encrypting the next data block of n records. The method of data encryption and transmission is repeated, **106**, until the entire data volume has been encrypted and transmitted.

The decryption method is depicted in FIG. **9** at the destination side, U_d. Initially, U_d receives a communication request from the CA, **108**. Then its DAK daemon forks a child communication process U_d_COM in order to freeze the generation of its version of DAK and sends the DAK_NRC to the CA for synchronization purposes, **110**. U_d_COM will run on behalf of the user until the end of the users' session communication. In the event of successful handshaking with the CA, **112** (FIG. **10**), the destination user receives an initial dynamic session key DSK, from the CA, **114**.

After successful handshaking, U_d uses the initial DSK to randomly generate n DSKs (DSK_i; $1 \leqq i \leqq n$) of the same size as the initial DSK (used as a seed), **116**. Beginning with the same initial DSK at both U_s and U_d sites, U_d randomly derives the same set of n regenerated DSKs as the source user U_s, and parallels the encryption method described at the source user side in reverse, to decrypt the transmitted data. U_d receives a block of n cipher records ($Cipher_i$; $1 \leqq i \leqq n$) **118**. All cipher records are decrypted using each record's corresponding DSK, **120**. Concatenation of the decrypted records provides the original message data to the destination user, **122**. U_d generates n new DSKs from the recovered block of data records and the current DSKs to be used for decryption of the next block of cipher records, **124**, as illustrated in FIG. **13**. The process is repeated, **126**, until all the transmitted cipher data has been decrypted.

Referring to FIG. **12**, a diagram illustrates the decryption method at **120** of FIG. **9** in greater detail. To shuffle the m bytes of each of the received n cipher records, **158**, back to the correct order, a permutation table PT_i is generated for each cipher record C_i, **160**. The permutation table is generated, as at the source side, as a copy of the corresponding DSK. Permutation is performed on the received cipher record bytes using each of the table entry's index and its associated entry value, as indices of the two bytes in the cipher record to be swapped, but in bottom-up order, **162**: permute $C_i[j]$ with $C_i[PT_i[j]]$, for j from m down to 1. The result is a temporary cipher record C_i^{tmp}, **164**. The decryption method continues by performing the XOR operation on $C_i^{tmp}[j]$, **164** and the corresponding $DSK_i[j]$, for $1 \leqq j \leqq m$ **166**. This results in the original data record D_i, for $1 \leqq i \leqq n$, **168**. The process continues as shown in FIG. **9**, at **122**.

As discussed in FIGS. **8** and **9**, upon request for a secure communication, both the source user and destination user must accomplish a successful handshaking process with the CA. FIG. **10** illustrates a user handshaking method with the CA. Upon receiving a message from the CA, **130**, the user responds accordingly, **132**. If an "abort" message is received, then the user terminates the communication child process in charge of the connection, **144**. If authentication is requested, **138**, the user mutually authenticates with the CA as described in FIG. **6**(*b*). If a "synchronize" message is received including a number (x), the user performs x regenerations of its DAK, in order to be synchronized with the CA, i.e., aligning to the same DAK **134**, as described in FIG. **5**. An acknowledgment message is then sent back to the CA, **136**.

After either synchronization or authentication, the user waits for a "session key" message including an encrypted DSK, (E(DSK)). The encrypted DSK is decrypted, **140**. Then, the communication child process returns the aligned DAK and the new DSK to its parent daemon in order to initialize a new state for the DAK regeneration state, **141**, and the communication between the source and destination is securely established, **142**. The process continues at **96** of FIG. **8** and **116** of FIG. **9**.

Turning to FIG. **13**, a diagram illustrates the dynamic session key (DSK) regeneration method of FIG. **8**, **102**, and FIG. **9**, **124**, where $DSK_i[j]$ represents the j^{th} byte of the last i^{th} dynamic session key, $D_i[j]$ represents the j^{th} byte of the last i^{th} data record, and $ExpK_i[h]$ represents the h^{th} byte of the i^{th} "expanded key", $1 \leqq i \leqq n$ and $1 \leqq j \leqq m$, $1 \leqq h \leqq 2m$. Each unit R represents the random selection of one byte among the $ExpK_i$'s 2m bytes. Each newly generated DSK_i is created using the last exchanged data record D_i, **172** and the last DSK_i, **170**. First, an expanded key ($ExpK_i$), **174**, of twice the size of DSK_i is generated. Each of DSK_i, D_i, and $ExpK_i$ is divided into (m/2) regions, indexed from 1 to (m/2). Each region of DSK_i

and D_i is of length 2 bytes, and each region of $ExpK_i$ is 4 bytes. The indexes of the four consecutive bytes of any region r, in the $ExpK_i$, are indexed with the values: 4r−3, 4r−2, 4r−1, and 4r. The indexes of the two consecutive bytes of any region r, in the DSK_i and D_i, are indexed with the values: 2r−1 and 2r. Filling the four bytes of region r in the $ExpK_i$, from DSK_i and D_i is defined as follows:

$$ExpK_i[4r-3] \leftarrow DSK_i[2r-1]$$

$$ExpK_i[4r-2] \leftarrow DSK_i[2r]$$

$$ExpK_i[4r-1] \leftarrow (DSK_i[2r-1])XOR(DSK_i[2r])$$

$$ExpK_i[4r] \leftarrow (DSK_i[2r-1])XOR(DSK_i[2r])XOR(D_i[2r-1])XOR(D_i[2r])$$

Then, a random selection of m bytes from the 2m bytes of $ExpK_i$ is performed, **178**, based on the $DSK_i[1]$ byte as the random function seed, **176**. Alternatively, any randomly selected byte can serve as the random function seed. This function generates a sequence of m random numbers, in the range between 1 and 2m, each of which represents the index of the byte to be placed as the next byte of the newly generated DSK_i, **180**. This operation is performed at both the source and destination user nodes. The method can be performed periodically or aperiodically (with mutual consent of source and destination), depending on the implementation of the invention.

It will be understood by those of skill in the art that the ExpK of FIGS. **13** and **14**, can alternatively be comprised of any number of bytes, and the invention is not limited to any particular size for ExpK. It will also be understood that the byte positions chosen to be XORed together to fill the positions of the ExpK of FIGS. **13** and **14** could similarly be altered, and still remain within the inventive scope of the dynamic encryption and authentication method. It will also be understood that the size of the regions of the keys used to regenerate new DSKs and DAKs, need not be 2 or 4 bytes, but could be of any size within the size of the respective keys. Similarly, alternative logic operations could be substituted for the XOR operation.

Although the invention has been described in detail with reference to this preferred embodiment, other embodiments can achieve the same results. Variations and modifications of the present invention will be obvious to those skilled in the art and it is intended to cover in the appended claims all such modifications and equivalents. The entire disclosures of all references, applications, patents, and publications cited above are hereby incorporated by reference.

What is claimed is:

1. A method of providing a secure data stream between system nodes, the method comprising:

providing a previous encryption key;

creating a data record at a source node, the data record including plaintext to be exchanged;

regenerating a new encryption key at the source node as a function of the data record and a previous encryption key by performing a logic operation on the previous encryption key and the data record; and

performing a logic operation on the previous encryption key and the data record to form an expanded key.

2. The method of claim **1** wherein the step of regenerating a new encryption key by performing a logic operation comprises regenerating a new encryption key by performing an XOR logic operation on the previous encryption key and the data record.

3. The method of claim **1** further comprising the step of selecting bytes from the expanded key to generate the new encryption key.

4. The method of claim **3** wherein the step of selecting bytes from the expanded key to generate the new encryption key comprises randomly selecting bytes from the expanded key to generate the new encryption key.

5. The method of claim **4** wherein the step of randomly selecting bytes from the expanded key to generate the new encryption key comprises randomly selecting bytes from the expanded key using a byte from the previous encryption key as a seed of random generation.

6. The method of claim **1** further comprising the step of encrypting the data record with the new encryption key to form encrypted data.

7. The method of claim **6** wherein the step of encrypting the data record with the new encryption key comprises performing a logic operation on the data record and the new encryption key.

8. The method of claim **7** wherein the step of performing a logic operation on the data record and the new encryption key comprises performing an XOR operation on the data record and the new encryption key.

9. The method of claim **7** wherein the step of performing a logic operation on the data record and the new encryption key comprises forming a cipher.

10. The method of claim **9** further comprising the step of permuting portions of the cipher to form another cipher.

11. The method of claim **6** further comprising the step of transmitting the encrypted data from the source node over a data stream.

12. The method of claim **11** further comprising the step of receiving the encrypted data at a destination node.

13. The method of claim **12** further comprising the step of decrypting the encrypted data at the destination node.

14. The method of claim **13** wherein the step of decrypting the encrypted data comprises decrypting with a previous decryption key.

15. The method of claim **14** further comprising the step of regenerating a new decryption key using the decrypted data and a previous decryption key.

* * * * *